Titre: Title:	Operational Knowledge Distillation for Efficient Testing and Enhancement of Autonomous Robotic Systems
Auteur: Author:	Dmytro Humeniuk
Date:	2026
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Humeniuk, D. (2026). Operational Knowledge Distillation for Efficient Testing and Enhancement of Autonomous Robotic Systems [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/76057/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/76057/
Directeurs de recherche: Advisors:	Foutse Khomh
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Operational Knowledge Distillation for Efficient Testing and Enhancement of Autonomous Robotic Systems

DMYTRO HUMENIUK
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Mars 2026

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Operational Knowledge Distillation for Efficient Testing and Enhancement of Autonomous Robotic Systems

présentée par **Dmytro HUMENIUK**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Gabriela NICOLESCU, présidente
Foutse KHOMH, membre et directeur de recherche
Jana PAVLASEK, membre
Lionel BRIAND, membre externe

DEDICATION

To my loved ones, who made this journey possible.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Prof. Foutse Khomh, for his continuous guidance, support, and encouragement throughout my PhD studies. His insights, high standards, and trust in my work played a crucial role in shaping this research and in my development as a researcher. I am also grateful to Prof. Mohammad Hamdaqa for his support during the later stages of my PhD, bringing new ideas and perspectives that helped enrich our research projects.

I would like to thank my research collaborators from our industrial partner Sycodal, in particular Houssem Ben Braiek and Thomas Reid, for their valuable discussions, technical contributions, and constructive feedback at various stages of this work. I am also thankful to the engineering team at Sycodal: Khaled, Maxime, Tarik, and Nicolas, for their practical insights and collaboration, which helped bridge research ideas with real-world industrial constraints.

I thank all my lab colleagues for the many enjoyable moments and insightful discussions we shared. I feel very fortunate and grateful to be surrounded by such colleagues, many of whom have become close friends.

I would also like to express my sincere gratitude to the Fonds de recherche du Québec – Nature et technologies (FRQNT) for their financial support during the later years of my PhD.

Finally, I thank my family and close friends for their patience, understanding, and unwavering support throughout this journey. Most importantly, I am deeply grateful to my spouse, Vaiana Moreau, for her constant encouragement and understanding. Without her, this journey would have been far more challenging.

RÉSUMÉ

Les systèmes robotiques autonomes modernes, tels que les voitures autonomes et les véhicules aériens sans pilote, sont hautement complexes et intègrent des composants matériels, logiciels et d'apprentissage automatique qui interagissent au sein d'environnements dynamiques et souvent imprévisibles. Avec l'avancement rapide de l'apprentissage automatique, un nombre croissant de composants basés sur l'apprentissage est intégré dans ces systèmes afin de percevoir leur environnement et de prendre des décisions au niveau du contrôle. Ces techniques peuvent apprendre des modèles utiles à partir de grandes quantités de données, permettant ainsi aux systèmes autonomes de fonctionner efficacement dans des environnements complexes et variables qui auraient été extrêmement difficiles, voire impossibles, à gérer avec des logiciels traditionnels fondés sur des règles.

Cependant, contrairement aux systèmes logiciels traditionnels, les composants basés sur l'apprentissage automatique ne reposent pas sur des règles explicites pouvant être vérifiées et testées de manière déterministe. Lorsque des entrées nouvelles ou inhabituelles, probablement non observées lors de l'entraînement, sont rencontrées, le risque de comportements indésirables ou dangereux augmente considérablement.

Afin d'assurer la sécurité de ces systèmes, des tests approfondis avant le déploiement dans le monde réel sont essentiels. En raison des limites des tests en conditions réelles, la simulation est couramment utilisée comme étape préliminaire d'évaluation, dans laquelle des défaillances sont recherchées en explorant des variations dans les environnements opérationnels. Dans ce contexte, les défaillances correspondent généralement à des événements critiques pour la sécurité, tels que des collisions, des manœuvres dangereuses ou des violations de contraintes par le système sous test. Cependant, les tests basés sur la simulation demeurent longs et coûteux en ressources en raison de l'immense espace de scénarios de test possibles et des variations environnementales. Par conséquent, maximiser l'efficacité et l'efficience de la génération de tests est essentiel. Bien que des travaux antérieurs aient démontré que les techniques de recherche et d'optimisation peuvent augmenter le nombre de défaillances révélées dans un budget de test donné, nous soutenons que la technique de recherche elle-même ne représente qu'une partie de la solution. Les connaissances opérationnelles jouent un rôle important dans la structuration du processus de test. Sans intégrer une compréhension du contexte opérationnel du système, les tests générés sont plus susceptibles de manquer des défaillances importantes et de présenter une pertinence réduite par rapport aux conditions réelles.

Dans la première partie de cette thèse, nous nous concentrons sur la génération de tests et proposons trois nouvelles approches exploitant les connaissances opérationnelles dans différentes phases du processus de recherche : la génération de la population initiale (RIGAA), l'amélioration de la représentation (RILaST) et la génération de comportements dynamiques valides (Dynasto). Dans la seconde partie, nous nous concentrons sur l'amélioration des systèmes guidée par les tests, en montrant comment des connaissances opérationnelles, telles que des données issues du monde réel, peuvent être combinées avec des tests basés sur la simulation afin d'améliorer les performances réelles des manipulateurs robotiques autonomes. Nous proposons deux approches supplémentaires et les évaluons dans un contexte industriel : l'amélioration du système réel guidée par les défaillances (MARTENS) et la réparation du système tenant compte de l'écart simulation-réalité (STRIM).

ABSTRACT

Modern autonomous robotic systems, such as self-driving cars and unmanned aerial vehicles, are highly complex, integrating hardware, software, and machine learning components that interact within dynamic and often unpredictable environments. With the rapid advancement of machine learning (ML), an increasing number of learning-based components are being incorporated into these systems to perceive their surroundings and make control-level decisions. These techniques can learn useful patterns from large amounts of data, allowing autonomous systems to operate effectively in complex and variable environments that would have been extremely difficult, if not impossible, to achieve with traditional rule-based software. However, unlike traditional software systems, ML-based components do not rely on explicit rules that can be deterministically verified and tested. When novel or unusual inputs, likely not seen during training, are encountered, the risk of undesirable or unsafe behavior increases substantially.

To ensure the safety of these systems, extensive testing prior to real-world deployment is essential. Due to the limitations of physical-world testing, simulation is commonly used as a preliminary evaluation stage, where failures are searched for by exploring variations in operational environments. In this context, failures typically correspond to safety-critical events such as collisions, unsafe maneuvers, or violations of system-level safety constraints by the system under test. However, simulation-based testing remains time-consuming and resource-intensive because of the vast space of possible test scenarios and environmental variations. Consequently, maximizing the effectiveness and efficiency of test generation is critical. While prior works have demonstrated that search and optimization techniques can increase the number of revealed failures within a given testing budget, we argue that the search technique itself represents only part of the solution. Operational knowledge plays an important role in shaping the testing process. Without incorporating an understanding of the system's operational context, the generated tests are more likely to miss important failures and show less relevance to the real-world conditions.

In the first part of the thesis, we focus on test generation and propose three novel approaches leveraging operational knowledge in different parts of the search process: initial population generation (RIGAA), representation improvement (RILaST) and valid dynamic behavior generation (Dynasto). In the second part, we focus on test-driven system improvement, demonstrating how operational knowledge, such as real-world data, can be combined with simulation-based testing to enhance the real-world performance of autonomous robotic

manipulators. We propose two additional approaches and evaluate them in an industrial context: failure-driven real-world system enhancement (MARTENS) and simulation-to-reality gap–aware system repair (STRIM).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AV	Autonomous Vehicle
ADAS	Advanced Driver Assistance System
ADS	Autonomous Driving System
ARS	Autonomous Robotic System
UAV	Unmanned Aerial Vehicle
RS	Robotic System
ML	Machine Learning
EA	Evolutionary Algorithm
ES	Evolutionary Search
GA	Genetic Algorithm
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
BO	Bayesian Optimization
CPS	Cyber-Physical System
DNN	Deep Neural Network
MOEA	Multi-Objective Evolutionary Algorithm
SUT	System Under Test
VAE	Variational Autoencoder

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Autonomous robotic systems (RS), such as autonomous vehicles, robots, and drones, are complex cyber-physical systems (CPS) that integrate hardware and software components to perceive their environment and perform planning and control in order to accomplish a given task [1]. Modern autonomous RS increasingly rely on machine learning–based components [2], whose behavior depends on the training data and can be difficult to predict when exposed to previously unseen inputs [3]. As a result, testing and validating such systems is a challenging task.

1.1 Research Context and Motivation

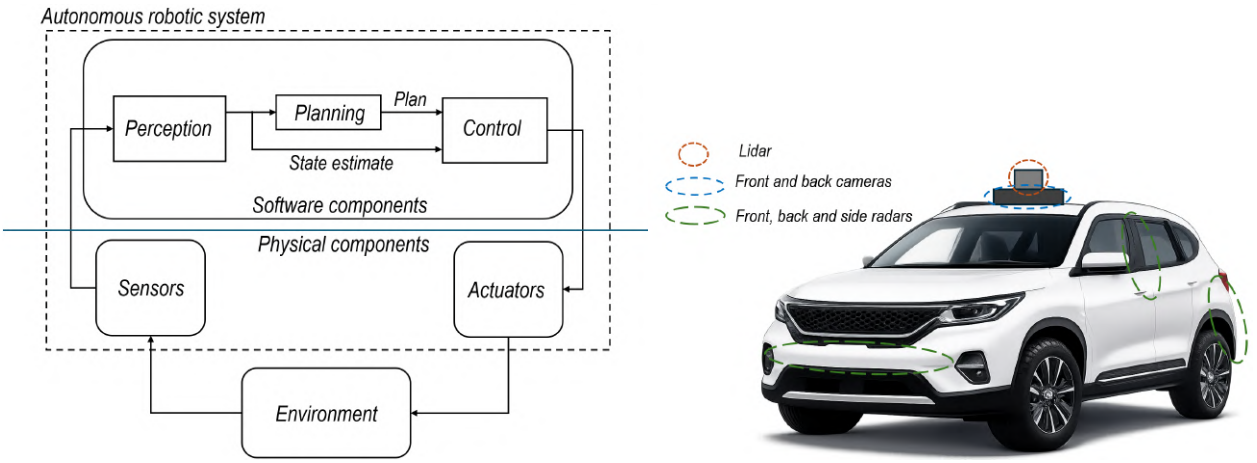
Autonomous robotic systems, such as self-driving cars, unmanned aerial vehicles, and mobile or manipulator robots, are software-driven systems that perceive and interact with the physical world through computer-controlled hardware components and sensors [4, 5]. Such systems promise significant societal and economic benefits. In the transportation domain, autonomous driving technologies aim to reduce traffic accidents caused by human error, improve mobility, and increase overall road safety. For example, a report by KPMG [6] estimates that widespread adoption of autonomous vehicles could reduce car accidents by up to 90% by 2050, potentially saving tens of thousands of lives annually, while also lowering the economic costs associated with collisions by more than 60%.

Beyond transportation, mobile robots and robotic manipulators can substantially improve efficiency and productivity in tasks traditionally performed by humans, while also enhancing safety in hazardous environments. Applications such as firefighting, mining, disaster response, and industrial automation benefit from deploying autonomous robots in settings that are dangerous, inaccessible, or physically demanding for human operators [7, 8].

At the same time, components enabling autonomy, such as perception modules, prediction models, behavior planning, and control algorithms, greatly increase system complexity, making the development and testing of such systems more challenging [9]. The overall diagram outlining the autonomous RS operating cycle is shown in Figure 1.1a. It consists of a continuous loop where the agent (i.e., the autonomous RS) interacts with the real world, i.e., the environment. The agent perceives its surrounding environment and estimates its position using various sensors, such as cameras, lidars and radars processed with data fusion algorithms [1]. It then uses a planning module to generate a trajectory towards a specified

goal. The control module produces commands that enable the agent to follow the trajectory, which are executed via the actuators, such as DC motors [10].

An example of a system with autonomy capabilities is shown in Figure 1.1b. It demostrates a vehicle equipped with a typical sensor suite for autonomous driving [11], highlighting key components involved in environmental perception and collision avoidance. The LiDAR unit provides high-resolution 3D mapping of the surroundings, which is used for object detection and classification as well as spatial awareness. Cameras support object classification, traffic signal recognition, and lane tracking. Typically, multiple cameras are installed in the vehicle, giving a 360 degree view of the surroundings. Radar sensors complement the data from LiDAR unit to build the environment representation and perform the surrounding object detection. Several radar units are installed in the front, back, and sides of the vehicle. Compared to the LiDAR unit, radars are less affected by adverse weather conditions such as rain, snow, or fog [12].



(a) Real-world operation cycle

(b) Example of an autonomous system

Figure 1.1 Autonomous robotic system operation diagram

Processing data from multiple onboard sensors and performing tasks such as localization and object detection requires complex perception and state-estimation algorithms. In recent years, learning-based approaches, including deep neural networks (DNNs), have demonstrated strong performance, particularly in complex and unstructured environments, and have become central components of modern autonomous systems [13, 14].

For planning and control, autonomous systems commonly rely on both traditional model-based techniques (e.g., optimization-based motion planning and control) and machine-learning-based approaches. While classical methods offer interpretability and safety guarantees,

learning-based planners and controllers enable more adaptive behavior in highly dynamic and uncertain environments, and are increasingly adopted in real-world deployments [15]. Notable examples include reinforcement-learning-based autonomous drones that outperform expert human pilots [16] or the TransFuser, a transformer-based autonomous driving model that won the 2022 CARLA Autonomous Driving Challenge [17].

Overall, state-of-the-art autonomous robotic systems increasingly rely on learning-based components for perception, decision making, or both. At the same time, ML-based components come with important limitations. A key limitation is their limited ability to generalize beyond the data distribution observed during training. ML-based systems are typically expected to operate under conditions similar to those encountered in their training data [18]. When this assumption is violated, the risk of undesirable or unsafe behaviors that falsify system requirements increases substantially. In the context of cyber-physical systems, these risks are combined with the complex software and hardware interactions, potentially leading to severe accidents. We present some concrete examples below.

In 2018, a fatal crash occurred in Arizona, USA, involving a developmental automated driving system (ADS) and a pedestrian [11]. The pedestrian, who was walking a bicycle, crossed a multi-lane road at night outside a designated crosswalk. The autonomous system repeatedly oscillated between classifying the detected object as a bicycle, a pedestrian, or an unknown object and failed to correctly predict the pedestrian's trajectory due to this classification instability. As a result, it did not appropriately reduce the vehicle's speed. As concluded by the investigation report, this misprediction likely arose from a rare combination of factors, including the unusual appearance of a pedestrian walking a bicycle and the pedestrian's presence in an atypical location, namely outside a crosswalk. More recently, in 2022, ADS failed to detect motorcycles on a highway in nighttime conditions, which resulted in fatal crashes [19], [20].

These examples confirm that edge cases and out-of-distribution events are unavoidable in real-world driving. It is therefore essential to conduct extensive testing of autonomous systems before deployment. However, real-world testing is both costly and time-consuming, and the occurrence of critical edge cases is rare [21]. As a result, simulation-based testing is oftentimes used as a preliminary evaluation stage, enabling controlled exposure to diverse scenarios, including rare and dangerous ones, without the risks associated with real-world trials [22], [23].

The diagram of simulation-based testing is shown in Figure 1.2. The operation cycle of autonomous RS is identical to the one from the real world. However, in this set-up the sensors and actuators are virtual, as well as the operating environment. The operating environment,

containing static and dynamic elements, is specified through a test scenario and it is the goal of a test generator to find challenging and failure revealing scenarios. Static elements remain invariant throughout the scenario execution and typically describe fixed environmental or configuration properties (e.g., road layout, object geometry or object placement). Dynamic elements, in contrast, evolve over time and capture entities or processes whose states change during execution (e.g., moving agents, or environmental dynamics). For example, in the case of an autonomous vehicle, static elements may include road topology, buildings, and the locations of fixed obstacles, while dynamic elements include pedestrians, other vehicles, and changing environmental conditions such as weather.

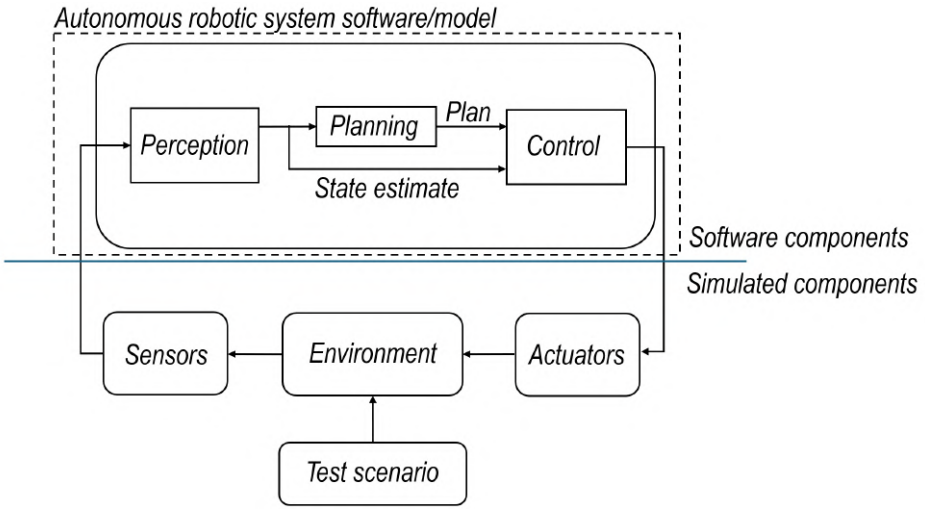


Figure 1.2 Simulation based testing of autonomous robotic systems

Given its importance, this thesis focuses on simulation-based testing, with particular emphasis on automated test scenario generation. In the following subsection, we outline the main limitations of simulation-based testing and the specific challenges addressed in this work.

1.2 Problem Statement and Research Objectives

As we have seen in one of the previous sections, modern ARS systems are highly complex, as they often involve the integration of hardware, software and machine learning components. Testing the system in the presence of these interacting components is a challenging problem. Simulation-based testing plays an important role in its solution, but still has its limitations, which we discuss below.

Computational efficiency. Simulation-based testing is less expensive than real-world testing, however, it still requires significant computational resources and time [24]. For instance,

one simulation of an autonomous vehicle can last from tens of seconds to minutes. Additionally, the behavior of the system under test can be non-deterministic, requiring multiple evaluations of the same scenario to establish a probabilistic metric of failure [25]. As a result, the practical applicability of simulation-based testing is constrained by available computational resources. This makes computational efficiency and sample-efficient test generation critical, as only a limited number of scenarios can be evaluated within allocated time and resource budgets.

Vast search space. A test scenario is typically defined by a high-dimensional combination of static parameters, such as environment layout, object placements, initial conditions, as well as dynamic elements. Safety-critical failures often occur only under rare and specific combinations of conditions, making them difficult to discover through random sampling or uniform exploration strategies [26], [27]. As a result, a large fraction of simulation effort may be spent evaluating uninformative or redundant scenarios, while failure-revealing cases remain sparsely distributed within the search space. This challenge motivates the need for intelligent, guided test generation approaches that can efficiently navigate the scenario space and focus exploration on regions that are most likely to expose system weaknesses.

Simulation to reality gap. Beyond the challenge of efficiently exploring a vast scenario space, simulation-based testing is further limited by the simulation-to-reality (sim2real) gap [28]. This gap becomes evident through the generation of unrealistic test scenarios, imperfections in system modeling within simulation, and a lack of photorealism. In this thesis, we focus on two major challenges related to sim2real. A first major challenge we consider is the difficulty of realistically modeling dynamic behaviors and interactions. Autonomous robotic systems often operate in environments with pedestrians, vehicles or other agents whose behavior is complex, context-dependent, and stochastic [29]. Simplified or overly aggressive behavior models, used in automated testing to provoke failures [30], may lead to unrealistic interactions that do not reflect plausible real-world dynamics. As a result, failures uncovered in simulation may stem from invalid or implausible agent behaviors rather than genuine system weaknesses, reducing the relevance of such test cases for real-world deployment. Ensuring behaviorally realistic and physically plausible dynamic interactions is therefore essential for simulation-based testing to produce meaningful and transferable insights.

A second major contributor to the sim2real gap that we consider concerns visual realism, which is particularly critical for autonomous systems relying on learning-based perception modules [31]. While modern simulators can generate large volumes of labeled synthetic data [32], synthetic imagery often differs from real-world sensor data in subtle but important

ways, including lighting conditions, textures, noise patterns, and optical artifacts. These discrepancies can lead to perception models that perform well in simulation but degrade significantly when deployed in real environments [33]. Increasing photorealism alone is often insufficient, as capturing the full diversity of real-world visual conditions remains challenging. Consequently, methods that explicitly account for visual domain differences are necessary to ensure that simulation-based testing and improvement of perception models generalizes to real-world operating conditions.

Real world system improvement. Beyond failure detection, an important and often overlooked challenge lies in translating simulation-based testing outcomes into tangible improvements of real-world autonomous robotic systems. In many existing workflows, failures uncovered in simulation are analyzed manually and treated as isolated artifacts, with limited systematic reuse for system enhancement [34].

In response to these challenges, this thesis aims to enhance the effectiveness and practical relevance of simulation-based testing for autonomous robotic systems. We argue that effective testing of autonomous robotic systems cannot be performed in isolation from their real-world operation context, and that incorporating operational knowledge into simulation-based test generation is essential for uncovering realistic failures and improving system robustness.

In this thesis, operational knowledge denotes information derived from, or informed by, the real-world operation of an autonomous robotic system. This includes real-world operational data, surrogate models that approximate system behavior under operational conditions, and knowledge about classes of inputs or situations that are likely to be challenging for the system. By incorporating the operational knowledge into the search process, we aim to achieve the following objectives:

1. **Effectively guide the test scenario search toward failure-revealing regions of the scenario space.** Existing techniques focus on new search algorithm configurations, search operators or simplified fitness functions, which are oftentimes problem specific [24, 25, 35] and may bias the search towards non-relevant search regions. In contrast, this thesis aims to propose search-level strategies for incorporating operational knowledge that can be applied on top of existing search-based testing algorithms. These strategies are designed to be applicable across a wide range of scenario generation problems and to promote the discovery of novel failure patterns, avoiding convergence around already known failures. In particular, we focus on the following parts of the search process:

   (a) Improvement of the search algorithm initialization.

(b) Improvement of the test scenario search space representation.

2. **Generate challenging and behaviorally realistic test scenarios.** The problem of behaviorally plausible scenario generation becomes prominent in dynamic test environments, in which parameters of entities, such as pedestrians or other moving agents, evolve over time [36]. Our objective is to leverage operational knowledge, such as real-world safety standards and guidelines, to generate behaviors for agents interacting with the system under test. These behaviors are designed to respect the defined constraints, ensuring that any discovered failures arise from plausible interactions.

3. **Bridge simulation-based testing and real-world system improvement.** The objective is to leverage failures discovered in simulation for improving the performance and robustness of real-world autonomous systems, while explicitly accounting for simulation-to-reality discrepancies in sensory observations.

1.3 Contributions and Impact

The main contributions of this thesis are illustrated in Figures 1.3 - 1.6 and summarized below. Overall, this thesis proposes five novel approaches for the testing and improvement of autonomous robotic systems, supported by open-source, reusable tools and datasets.

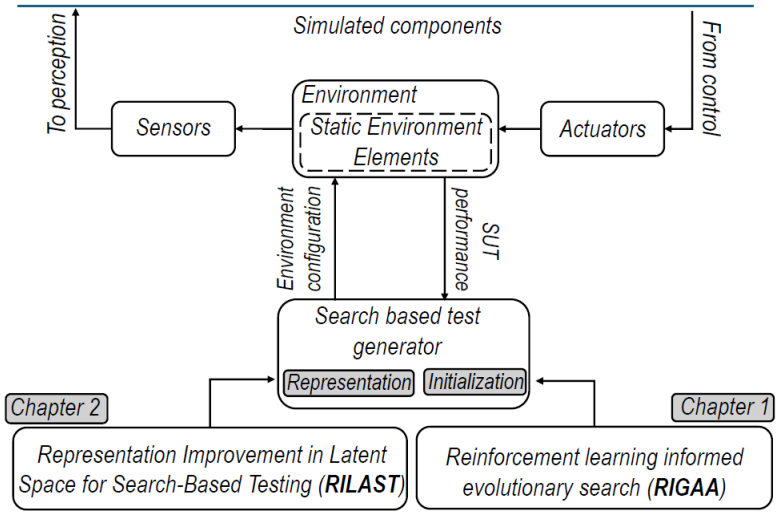


Figure 1.3 RIGAA and RILAST simplified diagrams

The first part of this thesis focuses on developing approaches for more efficient test generation in both static and dynamic environments. Three novel methods are proposed: **RIGAA**, **RILAST**, and **Dynasto**.

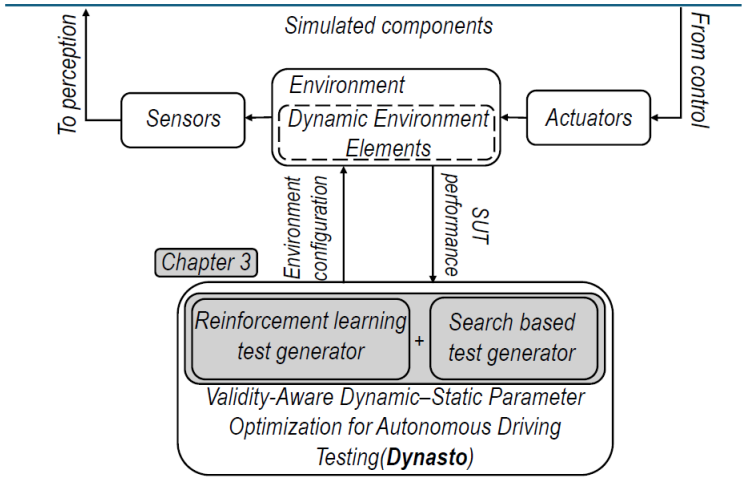


Figure 1.4 Dynasto approach simplified diagram

RIGAA (Reinforcement Learning–Informed Genetic Algorithm for Autonomous Systems Testing) integrates reinforcement learning into the evolutionary search process. A reinforcement learning agent learns useful problem constraints, which are then used to generate part of the genetic algorithm's initial population, thereby directing the search toward promising regions of the space from the beginning.

As observed in the experiments, the representation of test scenarios used within optimization algorithms has a significant impact on the search process. Given a limited testing budget, representation should be optimized to facilitate the search. To address the problem of representation improvement, we propose RILAST (Representation Improvement in Latent Space for Search Based Testing of Autonomous Robotic Systems) approach that leverages evolutionary search in conjunction with variational autoencoders to optimize the search over a latent space populated with challenging and diverse test scenarios.

The RIGAA and RILAST approaches primarily focus on static environmental elements, assuming the presence of a single agent within the environment. However, as environments become more dynamic, additional challenges emerge. To address these, we propose the DYNASTO framework (Figure 1.4, which targets the testing of autonomous driving agents in dynamic scenarios. DYNASTO combines evolutionary search with reinforcement learning to uncover a broader range of failures arising from both static initial conditions and dynamic interactions among entities in the simulated environment, while explicitly enforcing behavioral validity constraints to ensure realistic adversarial agent behavior.

In the second part of this thesis, the focus shifts to leveraging failures revealed in simulation to enhance the performance of the real-world system under test. The investigated problems

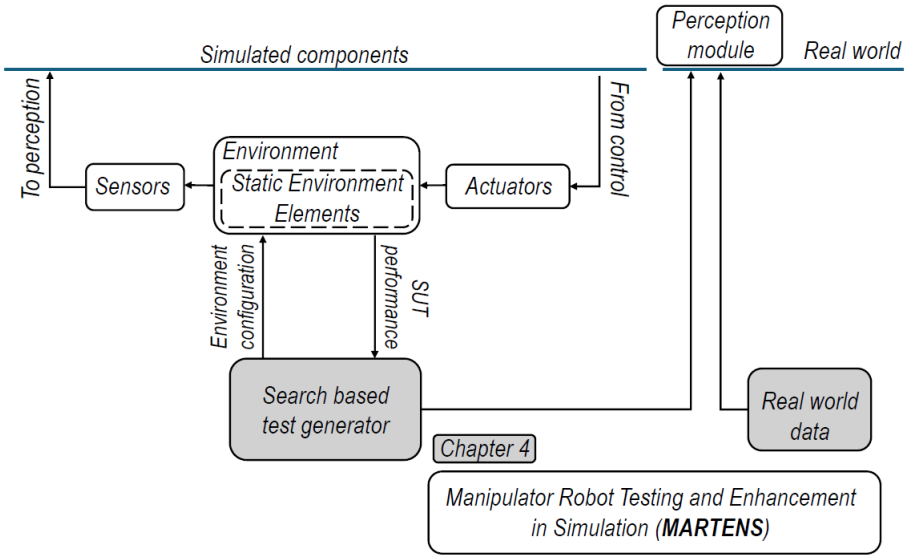


Figure 1.5 MARTENS approach simplified diagram

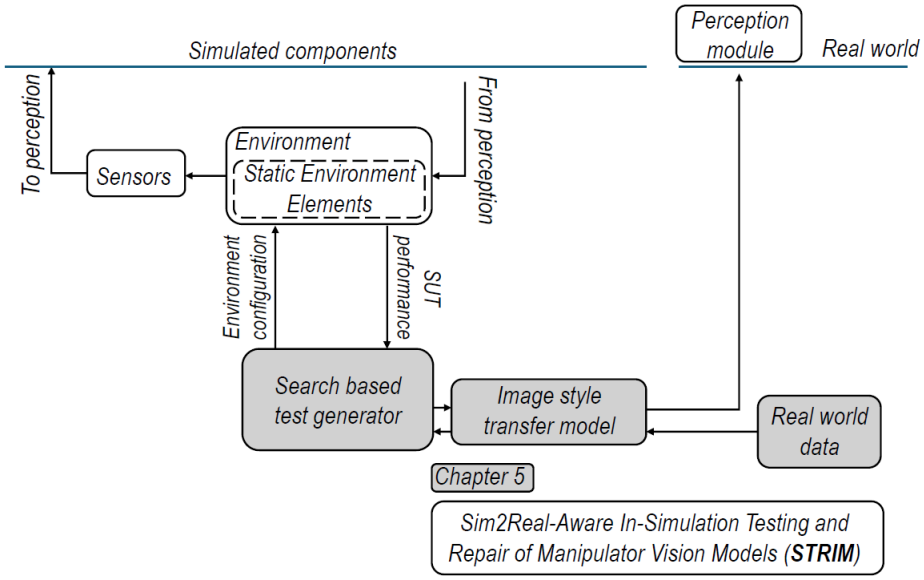


Figure 1.6 STRIM approach simplified diagram

originate from real-world use cases provided by our industrial partner, which specializes in the integration of robotic manipulators. We first propose **MARTENS** (Manipulator Robot Testing and Enhancement in Simulation) (Figure 1.5), a framework that integrates the photorealistic NVIDIA Isaac Sim simulator with evolutionary search to identify critical scenarios and combines them with real-world data to improve the performance of the physical system under test.

To further reduce the simulation-to-reality gap and minimize the need for manual annotation of real-world data, we propose **STRIM** (Sim2Real-Aware In-Simulation Testing and Repair of Manipulator Vision Models) (Figure 1.6). STRIM combines unlabeled real-world data with simulated data to train a generative image style transfer model that acts as a realism-enhancing filter for simulated images. This model is then integrated into the simulation to guide test generation toward failures that are more representative of real-world visual conditions. Finally, the real-world perception model is fine-tuned using failure cases revealed by STRIM, resulting in its improved performance.

The contributions of this thesis include a number of accepted and submitted publications, which are presented in Appendix A.

1.4 Thesis Outline

The rest of the proposal is organized as follows. In Chapter 2, we present a more detailed explanation of the concepts used in the proposed test scenario generation approaches, such as evolutionary search, reinforcement learning, generative models and perception guided robotic manipulators. In Chapter 3, we review the existing works related to testing autonomous robotic systems. Chapters 4 - 8 present the proposed RIGAA, RILAST, Dynasto, MARTENS and STRIM approaches respectively. Chapter 9 summarizes and concludes the thesis, and discusses the limitations and future work.

CHAPTER 2 BACKGROUND

In this section, we discuss the key concepts used throughout the thesis and the proposed contributions. Section 2.1 describes the evolutionary search (ES) process, which is used in all thesis contributions. In Section 2.2, we describe general reinforcement learning (RL) concepts, which are important for the contributions presented in Chapter 4 and Chapter 6. Section 2.3 describes autoencoder-based generative models used in Chapter 5.3, while Section 2.4 describes style-transfer generative models utilized in Chapter 8. Section 2.5 provides background on robotic manipulators and object detection models, representing the family of systems studied in Chapters 7 and 8. Finally, Section 2.6 defines the concept of operational knowledge used throughout the thesis.

2.1 Evolutionary Search

Evolutionary algorithms (EAs), such as genetic algorithms, are commonly used in search-based software testing [37–39]. We describe these methods in more detail below. Evolutionary algorithms are a family of optimization algorithms that are inspired by the mechanisms of biological evolution, such as reproduction, mutation, and selection. The goal of EAs is to search for the optimal solution(s) of a given problem by iteratively generating and evaluating a population of candidate solutions, and applying the aforementioned mechanisms to evolve the population towards better solutions [40].

Given a particular problem, we start by generating a collection of candidate solutions. Their quality or fitness (that is, how well they solve the problem) determines the chance that they will be selected and used as seeds for constructing further candidate solutions through recombination and mutation. The new individuals have their fitness evaluated and compete for survival. Over time such 'natural' selection mechanism causes a rise in the fitness of the population. The general EA process is illustrated in Figure 2.1.

Genetic algorithms (GA) are a class of evolutionary algorithms that use a set of principles from genetics, such as selection, crossover, and mutation, to evolve a population of candidate solutions [41]. A typical genetic algorithm pipeline is shown in Fig. 2.2. The basic idea is to start with a set of individuals (candidate solutions) representing the initial population, usually generated randomly from the allowable range of values. Each individual is encoded in a specific representation, such as a bit string or a vector of integer or real-valued parameters, and is referred to as a chromosome. A chromosome is composed of genes. Each individual is

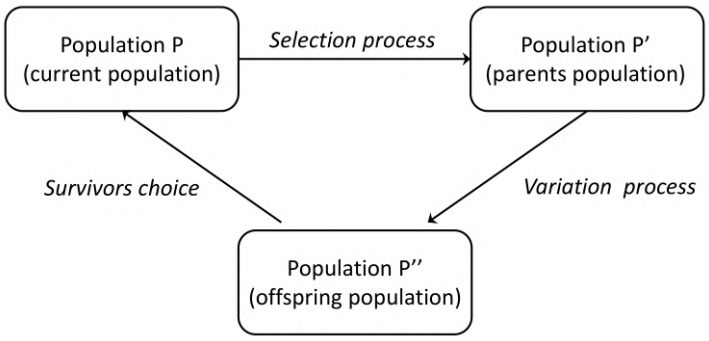


Figure 2.1 General functioning of evolutionary algorithms

evaluated and assigned a fitness value. Some of the individuals are selected for mating and undergo genetic operations, including crossover and mutation. The search is continued until a stopping criterion is satisfied or the number of iterations exceeds a specified limit.

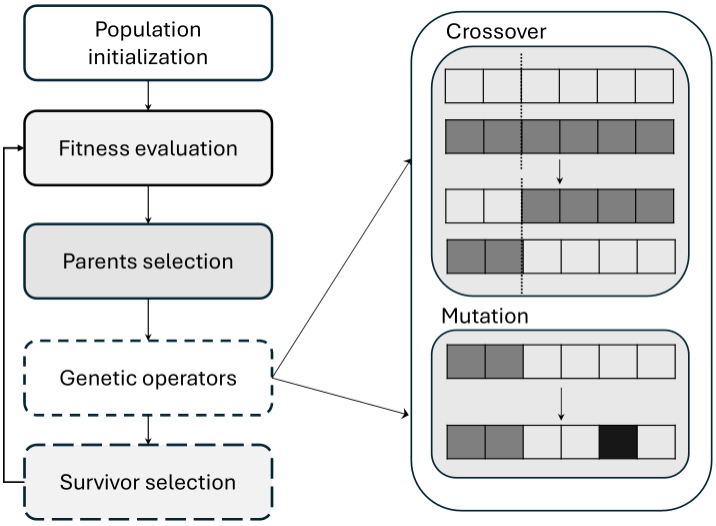


Figure 2.2 Genetic algorithm pipeline

Three genetic operators are used to evolve the solutions: selection of survivors and parents, crossover, and mutation. The operators in evolutionary algorithms serve different functions. Mutation and crossover operators promote diversity in the population, allowing for the exploration of the solution space. On the other hand, survivor and parent selection operators promote the quality of the solutions, enabling the exploitation of the search space. It is essential to select the appropriate combination of operators that offers a good balance between exploration and exploitation to achieve optimal results.

Parents Selection. Selection of parents is an operator that gives solutions with higher fitness

a higher probability of contributing to one or more children in the succeeding generation. The intuition is to give better individuals more opportunities to produce offsprings. One of the commonly used selection operators is tournament selection [42]: a small subset of individuals is chosen at random, and then the best n individuals in this set are selected for the mating. The selection pressure can be adjusted by controlling the size of the subset used. Another commonly used technique is a proportional-based selection, also known as a roulette wheel selection, where the probability of choosing an individual depends directly on its fitness.

Crossover operator. The crossover operator is used to exchange characteristics of candidate solutions among themselves. In our approach, we are using a one-point crossover, illustrated in Fig. 2.3. We can see how the parent chromosomes are exchanged genes t_i to produce the offsprings. Other types of crossover include multi point crossover and uniform crossover, however, they are only applicable to fixed-size individuals.

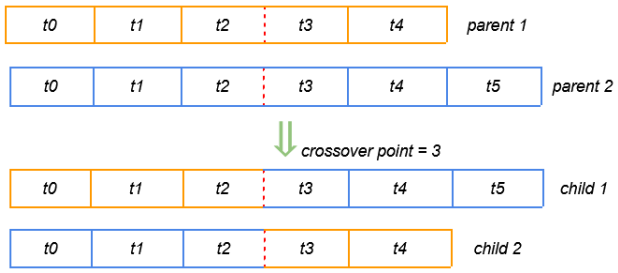


Figure 2.3 One point crossover operator

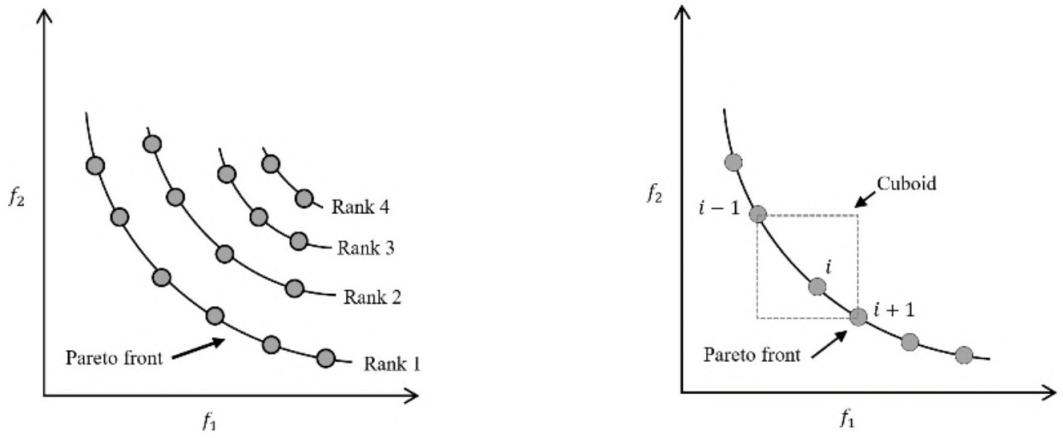
Mutation operator. The mutation operator has been introduced to prevent convergence to local optima; it randomly modifies an individual's genome (e.g., by flipping some of its bits, if the genome is represented by a bitstring) [43].

Crossover and mutation are performed with probability *pcross* and *pmut* respectively, where $pmut < pcross$. The rates at which mutation and crossover are applied are an implementation decision.

Survivors selection. In GA the population size is almost always constant. This requires a choice to be made about which individuals will be allowed into the next generation. This decision is often based on their fitness values, favoring those with higher quality, although the concept of age is also frequently used. In contrast to parent selection, which is typically stochastic, survivor selection is often deterministic. For genetic algorithms $(\mu + \lambda)$ strategy is commonly used. In this strategy, the set of offspring and parents are merged and ranked according to (estimated) fitness, then the top μ individuals are kept to form the next generation. [40].

The choice of optimal representation and operators for a genetic algorithm is problem dependent and is a complex area of research [44].

Genetic algorithms are effective in solving both single and multi-objective optimization problems [45]. However, they are particularly suited to multi-objective problems since they can simultaneously handle a set of possible solutions, allowing the algorithm to find multiple members of the Pareto optimal set in a single run. Under Pareto optimality, a solution is better than another if it is superior in at least one of the individual fitness functions and not worse in any of the others. This is an alternative to simply aggregating fitness using a weighted sum of the n fitness functions [40]. When using Pareto optimality to search for solutions, the result is a set of non-dominated solutions. Each solution in the non-dominated set is no worse than any other solution in the set, but it cannot be said to be better either. In a multi-objective genetic algorithm, the quality of a solution is determined by its non-dominance ranking, rather than by a single fitness value. An example of non-dominated sorted individuals is shown in Fig.2.4a (lower rank is better) [46].



(a) Non-dominated ranking

(b) Crowding distance calculation

Figure 2.4 Non-dominated sorting procedure and Crowding distance calculation

One is the most popular multi-objective genetic algorithms is the non-dominated sorting algorithm-II (NSGA-II) [47]. It builds a population of competing individuals, ranks and sorts each individual according to a non-dominated level, applies evolutionary search operators to create new pool of offsprings, and then combines the parents and offspring before partitioning the new combined pool into fronts. The NSGA-II then assigns a crowding distance to each member. The crowding distance value of a particular individual is the average distance to its two neighboring individuals in the Pareto front. The crowded distance of the *ith* solution is the average side-length of the cuboid, as shown in Fig.2.4b. It uses the crowding distance in

its selection operator to keep a diverse front by making sure each member stays a crowding distance apart. This keeps the population diverse and helps the algorithm to explore the fitness landscape [45].

2.2 Reinforcement Learning

Reinforcement learning is a subfield of machine learning that focuses on training agents to make sequential decisions in an environment in order to maximize a reward signal. RL algorithms learn through trial and error, interacting with the environment and adjusting their behavior based on the received rewards [48].

In the standard mathematical formulation of an RL problem, an RL agent learns to take actions in an environment in order to maximize a reward signal.

This is modeled as a Markov decision process (MDP), which is defined by a 5-tuple: S: a set of states that the agent can occupy, A: a set of actions that the agent can take, $R : S \times A \to \mathbb{R}$: a reward function that maps state-action pairs to real-valued rewards, $P : S \times A \to S$: a transition function that defines the probability of transitioning to a new state given the current state and action, $\rho_0 : S \to [0, 1]$: an initial state distribution that specifies the probability of starting in each state.

Typically, an RL agent interacts with an environment for one episode, and then its state is reset. An episode is a sequence of interactions between an agent and its environment, starting from an initial state and proceeding through a series of state transitions based on the actions taken by the agent. An episode typically concludes when the agent reaches a terminal state or when a specified time limit is reached. The RL agent's goal is to take actions that maximize the expected sum of future rewards obtained during an episode.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

where R_t is the expected reward at time t, r_{t+k+1} is the reward obtained at time $t + k + 1$, and γ is the discount factor that determines the importance of future rewards.

Value functions. In reinforcement learning, value functions quantify how desirable states and actions are with respect to a given policy. Two fundamental value functions are commonly used: the state-value function and the action-value function.

The state-value function $V^\pi(s)$ represents the expected cumulative discounted reward ob-

tained when the agent starts in state s and follows policy π thereafter. It is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t \mid s_t = s\right], \tag{2.2}$$

Intuitively, $V^{\pi}(s)$ measures how good it is for the agent to be in state s, assuming it continues to act according to policy π.

The action-value function $Q^{\pi}(s, a)$ represents the expected cumulative discounted reward obtained by taking action a in state s and then following policy π. It is defined as:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[R_t \mid s_t = s, a_t = a\right]. \tag{2.3}$$

While $V^{\pi}(s)$ evaluates the quality of a state, $Q^{\pi}(s, a)$ evaluates the quality of a specific state–action pair.

The optimal state-value and action-value functions correspond to the maximum expected return achievable by any policy:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), \tag{2.4}$$

$$Q^{*}(s, a) = \max_{\pi} Q^{\pi}(s, a). \tag{2.5}$$

Learning or approximating these optimal value functions is a central objective of many reinforcement learning algorithms.

A fundamental challenge in reinforcement learning is credit assignment: rewards may be delayed, and the agent must determine which earlier actions were responsible for future outcomes. Computing the full return R_t requires waiting until the end of an episode, which can be inefficient or impractical in long-horizon problems.

Temporal-difference learning. Temporal-difference (TD) learning addresses this challenge by allowing agents to learn incrementally from incomplete episodes. TD methods update value estimates using bootstrapping, i.e., by relying on their own current estimates of future value rather than waiting for the final return.

In the temporal-difference update, $V(s_t)$ represents the agent's current estimate of the expected cumulative discounted reward when starting from the current state s_t and following policy π. It reflects the agent's belief about the long-term desirability of the current state based on previously observed experience.

The term $r_{t+1} + \gamma V(s_{t+1})$ corresponds to a one-step estimate of the return. It combines

the immediate reward r_{t+1} received after taking an action in state s_t with the discounted estimated value of the next state s_{t+1}. This quantity serves as a target toward which the value estimate $V(s_t)$ is updated.

The difference between these two quantities,

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.6}$$

is known as the temporal-difference error and measures how much the observed outcome deviates from the agent's prior expectation.

For a state-value function $V(s)$, a one-step TD update is given by:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right], \tag{2.7}$$

where α is the learning rate and the term in brackets is known as the temporal-difference error. TD learning forms the basis of many modern RL algorithms due to its ability to learn online without requiring a model of the environment.

RL algorithms can be broadly categorized into off-policy and on-policy methods.

Off-policy methods. An RL algorithm is said to be off-policy if it learns the value of a target policy while following a different behavior policy. Q-learning is a canonical off-policy TD algorithm that directly estimates the optimal action-value function $Q^*(s, a)$ [48].

The Q-learning update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]. \tag{2.8}$$

Here r_{t+1} is the the immediate reward received after taking an action in state s_t and $\gamma \max_{a'} Q(s_{t+1}, a')$ is the discounted estimate of the maximum future return achievable from the next state s_{t+1}. The maximization over actions a' reflects the assumption that the agent will behave optimally from the next state onward. Throughout the learning process, these action-value estimates $Q(s, a)$ are iteratively updated and stored in a tabular form, commonly referred to as a Q-table, which is refined as more state–action transitions are observed.

In problems with large or continuous state spaces, storing $Q(s, a)$ in a table becomes infeasible. Deep Q-Networks (DQN) address this limitation by approximating the Q-function with a neural network [49]:

$$Q(s, a; \theta) \approx Q^*(s, a), \tag{2.9}$$

where θ denotes the network parameters. DQN combines TD learning with experience replay

and target networks to stabilize training and is among the most commonly used reinforcement learning algorithms [50].

On-policy reinforcement learning algorithms learn exclusively from data generated by the current policy. Unlike value-based methods, which learn how good actions are, policy gradient methods directly learn how the agent should act by optimizing the parameters of the policy itself.

On-policy methods. A policy $\pi_\theta(a|s)$ is typically represented by a parameterized function, such as a neural network, with parameters θ. Policy gradient methods adjust these parameters so as to increase the expected cumulative reward. At a high level, the policy parameters are updated in the direction that increases the likelihood of actions that led to higher rewards:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta), \tag{2.10}$$

where $J(\theta)$ denotes the expected return under the current policy and α is the learning rate.

In practice, the gradient $\nabla_\theta J(\theta)$ is estimated from sampled trajectories and encourages actions that resulted in high returns to become more likely in the future. This family of methods is referred to as policy gradient algorithms.

To reduce variance and improve learning stability, policy gradient methods often rely on a value function to assess how good an action was compared to an average behavior. This difference is captured by the advantage function:

$$A(s_t, a_t) = R_t - V(s_t), \tag{2.11}$$

which measures whether an action performed better or worse than expected in a given state.

Proximal Policy Optimization (PPO) [51] is a widely used on-policy actor–critic algorithm that builds upon policy gradients. PPO updates the policy using sampled data while restricting how much the policy is allowed to change at each iteration in order to improve training stability. Rather than directly maximizing the return, PPO maximizes a clipped objective that penalizes overly large policy updates. This clipping mechanism prevents destructive updates and makes PPO particularly robust in practice.

2.3 Generative Models for Representation Learning

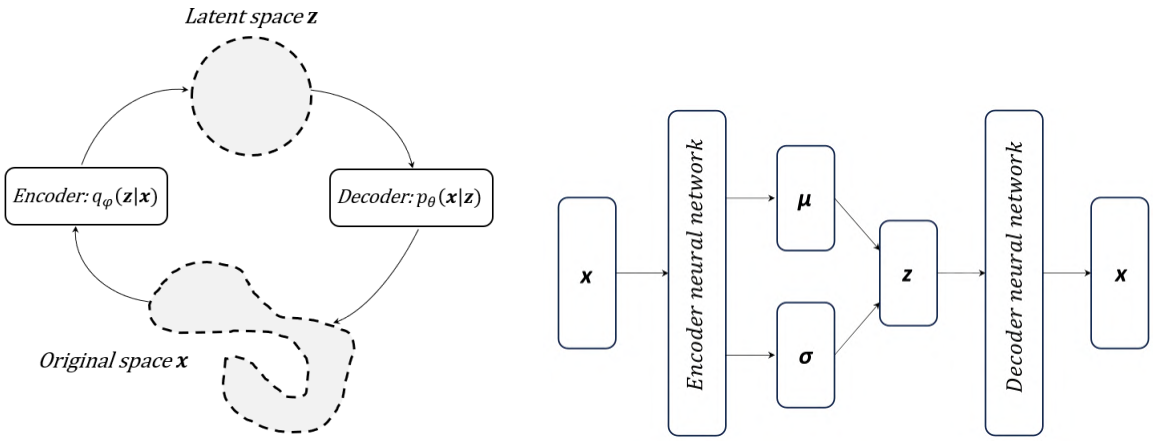
Generative models, such as autoencoders, generative adversarial networks (GANs), and variational autoencoders (VAE), rely on latent variables sampled from a latent space to learn

and approximate the original data distribution [52, 53]. By sampling from this distribution, they can generate new data similar to the original dataset. In other words, they learn a probabilistic representation of the input data. At the same time, GANs and autoencoders have a different underlying training methodology. In GANs, two modules need to be trained, namely the generator and the discriminator, which have opposite goals. The generator should learn to produce data that the discriminator cannot differentiate from the original data distribution. Because of complex training dynamics, GANs have well-known challenges in training, such as mode collapse and instability [54]. Autoencoders, by contrast, optimize a well-defined objective function by minimizing the reconstruction loss between input and output data, which enhances their stability and makes them easier to train [55].

Autoencoders [56] encode the input into a fixed latent vector z, which is more suitable for data compression than for generation. During training, they are designed to minimize the reconstruction loss (e.g., mean squared error or binary cross-entropy), which is effective for memorizing the inputs rather than learning a useful and generalizable representation of the data. This limitation makes vanilla autoencoders less suitable for our task of representation improvement.

At the same time, variational autoencoders [53] learn a probabilistic distribution over the latent space, which enables better generalization and meaningful data interpolation. Unlike vanilla autoencoders, which can overfit to the training data, VAEs introduce a regularization term that encourages smooth and continuous latent representations. During training, they minimize a combined loss: a reconstruction loss to ensure the decoded outputs resemble the inputs and a Kullback-Leibler (KL) divergence term to regularize the latent space, preventing overfitting and promoting meaningful structure in the learned representation. This makes VAEs better suited for representation improvement tasks. Therefore, we chose VAEs for further experiments in the proposed RILAST approach.

As shown in Figure 2.5a, the architecture of a VAE consists of two main components: encoder and decoder. Let x denote an input data sample from the dataset (e.g., an image or feature vector), and let z denote the corresponding latent variable representing a compressed representation of x in the latent space. The encoder learns a distribution $q_\phi(z|x)$ that approximates the true posterior $p(z|x)$, mapping data points into a structured latent space, where ϕ denotes the learnable parameters of the encoder network. This latent space captures the essential variations in the data, allowing for smooth interpolations and meaningful representations. The decoder, defined by $p_\theta(x|z)$, where θ denotes the learnable parameters of the decoder network, learns to generate realistic outputs from latent samples. Training a VAE involves optimizing the Evidence Lower Bound (ELBO), which balances two objectives.



(a) VAE concept diagram (b) VAE implementation diagram

Figure 2.5 VAE diagrams

The first term ensures that the decoder can accurately reconstruct the input, encouraging meaningful representations in the latent space. The second term regularizes the latent distribution by enforcing it to be close to a prior distribution $p(z)$, typically a standard normal distribution $\mathcal{N}(0, I)$. This prevents overfitting and ensures smoothness in the latent space. The ELBO loss function is given by:

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi(z|x)} \left[\log p_\theta(x|z)\right] - D_{\mathrm{KL}}(q_\phi(z|x) \parallel p(z)) \tag{2.12}$$

The first term represents the expected log-likelihood of reconstructing x from the latent variable z, where z is sampled from the learned latent distribution $q_\phi(z \mid x)$. This term acts as a reconstruction loss, encouraging the model to generate an output that closely matches the original input x_i from its latent representation z_i. In practice, this term is often computed using the Mean Squared Error (MSE) for continuous data or Binary Cross-Entropy (BCE) for binary data. The second term is the KL divergence, which encourages the learned latent distribution $q_\phi(z|x)$ to be close to the prior $p(z)$.

Figure 2.5b illustrates the architecture of a VAE, including its key components. The process begins with an input x, which is passed through an encoder, which is typically implemented as a neural network (NN). The encoder maps x to a latent distribution, parameterized by a mean μ and a standard deviation σ, rather than directly to a fixed latent vector. The latent variable z is then sampled from this distribution. The sampled latent representation z is then passed through the decoder, which reconstructs an approximation of the original input

x.

2.4 Generative Style Transfer Models

Here, we discuss another family of generative models, namely, generative style transfer models. These models aim to translate images from a source domain to a target domain while preserving the underlying semantic content. Both supervised and unsupervised image-to-image translation approaches exist. Supervised methods, such as pix2pix [57], learn the translation from paired source-target image examples, which enables accurate and controllable mappings but requires aligned datasets that are often difficult to obtain in practice. In contrast, many style transfer methods operate in an unpaired setting, where corresponding source-target image pairs are unavailable. This property makes unpaired approaches particularly attractive for simulation-to-real (sim-to-real) applications, where collecting aligned real-world data is often impractical or costly [33, 58]. We thus adopt an unpaired style transfer strategy in the proposed STRIM approach. Since no paired supervision is available, additional constraints must be introduced to avoid irrelevant solutions and ensure meaningful translations [59]. Below, we review some of the established unpaired style transfer strategies.

Neural Style Transfer. One of the first approaches to perform style transfer in images is the Neural Style Transfer (NST) approach that aims to combine the content of one image with the artistic style of another. The work by Gatys et al. [60] formulates style transfer as an optimization problem over pixel values, where a pretrained convolutional neural network is used to extract content representations from deep layers and style representations from the Gram matrices [61] of feature activations in shallow layers. However, it can be challenging to correctly distinguish style and content responsible pixels. Moreover, this approach operates on individual image pairs and does not learn a domain-level mapping, which restricts its applicability, since a single image cannot capture the full diversity of objects and configurations encountered in robotic settings. As a result, dataset-level translation models are required.

CycleGANs. One of the earlier approaches to unpaired style transfer is CycleGAN, introduced by Zhu et al. [59]. It simultaneously trains two generative adversarial networks (GANs) by learning two generators, $G : X \to Y$ and $F : Y \to X$, where X and Y denote two image domains representing different visual styles (e.g., simulated and real-world images). These generators are trained along with two corresponding adversarial discriminators, D_Y and D_X. The adversarial losses encourage the translated images to match the distributions of the target domains, while the cycle-consistency loss enforces that translating an image to

the target domain and back should reconstruct the original image:

$$F(G(x)) \approx x, \quad G(F(y)) \approx y.$$

This formulation is motivated by the intuition that a meaningful style translation should be cycle-consistent. For example, if a sentence is translated from English to French and subsequently translated back from French to English, the result should closely match the original sentence. This approach was subsequently adopted in robotics [62, 63] as well as in the cyber-physical systems (CPS) testing literature [28, 64]. However, CycleGANs have several limitations, including the need to train two generators and two discriminators, which increases computational cost, as well as the inherent instability of GAN training, which may lead to convergence issues or the generation of visual artifacts.

Contrastive Unpaired Translation. Contrastive Unpaired Translation (CUT) [65] is a more recent approach that simplifies unpaired image-to-image translation by using a single generator and a single discriminator. Instead of relying on cycle consistency, CUT introduces a patch-wise contrastive learning objective that encourages content preservation. Specifically, the method maximizes the mutual information between corresponding patches of the input image and its translated output at multiple feature layers within the generator network.

This contrastive constraint promotes the preservation of spatial and semantic structure, ensuring that local regions in the translated image remain aligned with the input, while still allowing the global appearance to adapt to the target domain. By eliminating the reverse generator required in CycleGAN, CUT reduces model complexity and training time, while achieving comparable or superior translation quality. The authors further propose a computationally more efficient variant, referred to as FasterCUT, which accelerates training by simplifying the contrastive loss computation. To the best of our knowledge, CUT type models have not been yet applied in the CPS testing literature.

2.5 Vision-Guided Robotic Manipulators

Vision-guided robotic manipulators are robotic arms equipped with visual perception systems that allow them to interact intelligently with their environment [5]. These systems combine mechanical manipulators, sensors (e.g., cameras or depth sensors), and control algorithms to perform tasks such as object picking, sorting, assembly, or human-robot collaboration. Vision guidance enables the manipulator to perceive objects, estimate their pose, and plan motions that are adaptive to dynamic and unstructured environments. Figure 2.6 shows an example of a robotic manipulator in a virtual industrial setting, which we used as one of the

use cases.

Figure 2.6 Example of a robotic manipulator in a simulated environment

A critical capability of vision-guided manipulators is object detection, which involves identifying and localizing objects of interest within an image. Formally, given an input image $I \in \mathbb{R}^{H \times W \times C}$, an object detector predicts a set of bounding boxes

$$\mathcal{B} = \{(b_i, c_i, s_i)\}_{i=1}^{N},$$

where $b_i = (x_i, y_i, w_i, h_i, r_i)$ denotes the bounding box center coordinates, width, height and rotation, c_i is the predicted object class, and $s_i \in [0, 1]$ is a confidence score indicating the likelihood that the box contains an object of class c_i [66].

Among modern detectors, YOLO (You Only Look Once) models are widely used in robotic manipulation due to their speed and efficiency [67]. YOLO formulates object detection as a single end-to-end regression problem. Rather than relying on sequential stages, it learns a function

$$f_\theta : I \to \mathcal{B},$$

parameterized by neural network weights θ, that maps the entire input image directly to bounding box coordinates and class probabilities in a single forward pass. This unified approach allows YOLO to exploit global contextual information and achieve low-latency inference, which is critical for real-time robotic control.

YOLO is classified as a single-stage detector, directly predicting object locations and classes, in contrast to two-stage detectors such as R-CNN-based models [68]. Two-stage detectors

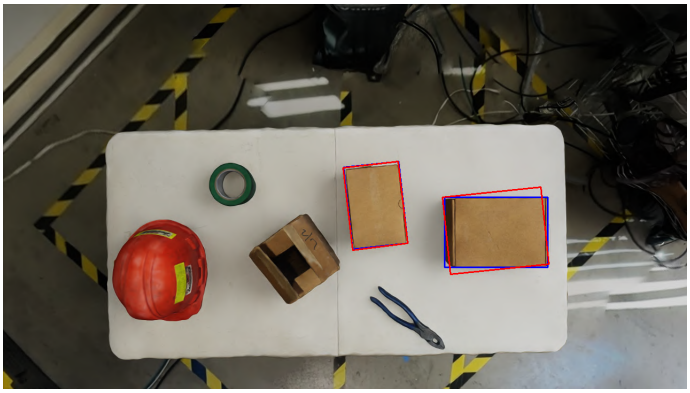


Figure 2.7 Example of object detection output used for manipulator grasp planning.

first generate a set of region proposals

$$\mathcal{R} = \{r_j\}_{j=1}^{M},$$

and then classify and refine each region. While precise, this approach increases computational cost and pipeline complexity. Single-stage detectors like YOLO answer both "where are the objects?" and "what are they?" in a single pass, enabling faster inference suitable for vision-guided robotic manipulators operating in dynamic environments.

In our work, specifically in the MARTENS and STRIM approaches, we leverage YOLO-based object detection models. Figure 2.7 shows an example of the predicted oriented bounding boxes (in red) alongside the ground truth boxes (in blue).

2.6 Operational Knowledge Definition

We define operational knowledge as structured information that characterizes how an autonomous robotic system behaves, should behave, or has behaved under specific operating conditions. Operational knowledge provides a basis for assessing the validity, relevance, or criticality of test scenarios and system executions within a testing process. In this thesis, occasionally, we use the term 'operational' and 'domain' knowledge interchangeably.

Formally, operational knowledge can be expressed through one or more functions defined over the space of test scenarios. In particular, it can be modeled as an evaluation function

$$\kappa : \mathcal{S} \to \mathbb{R} \text{ or } \{0, 1\}, \tag{2.13}$$

where $\mathcal{S}$ denotes the space of test scenarios parameterized by static and dynamic elements,

and $\kappa(s)$ quantifies whether a scenario $s \in \mathcal{S}$ is challenging, failure-revealing, or otherwise relevant for testing. The output may be binary (e.g., challenging vs. non-challenging) or continuous, reflecting degrees of criticality, risk, or relevance.

Operational knowledge may also encode domain-specific rules and constraints, such as traffic laws or safety requirements. In this case, it can be formalized as a rule-checking function

$$\rho : \mathcal{S} \to \{\text{valid}, \text{invalid}\}, \tag{2.14}$$

which determines whether a given scenario or an associated system execution complies with predefined operational rules.

Finally, operational knowledge may be derived from real-world operational data, such as logs, sensor recordings, or historical execution traces. In some settings, the knowledge is extracted through a dedicated transformation function:

$$\mathcal{E} : \mathcal{D} \to \mathcal{K}, \tag{2.15}$$

where $\mathcal{D}$ represents real-world data and $\mathcal{K}$ denotes the space of operational knowledge abstractions. The function $\mathcal{E}$ extracts latent knowledge in the form of priors, distributions, heuristics, or learned models.

Overall, operational knowledge provides a unifying abstraction for integrating expert rules, learned behaviors, and real-world evidence into the automated testing and validation of autonomous robotic systems.

In this thesis, we propose approaches that leverage operational knowledge to improve the relevance and effectiveness of simulation-based test scenarios.

CHAPTER 3 LITERATURE REVIEW

In this chapter, we review state-of-the-art test scenario generation approaches addressing the challenges outlined in Section 1. Autonomous robotic systems represent complex cyber-physical systems. The problem of identifying critical scenarios for cyber-physical systems has been addressed in several previous works on falsifying temporal logic requirements of cyber-physical systems, such as S-Taliro [69], Breach [70], and ARIsTEO [71]. These works focus on falsifying a model of the system that maps a set of input signals to a corresponding set of output signals.

In our review, we focus on testing autonomous RS for which the input signals are complex and may include data from various sensors and cameras. Generating a valid combination of falsifying input signals (such as lidar readings and RGB camera readings) directly would be unrealistic and impractical. Therefore, we focus on test scenario generation approaches that specify semantic aspects of the virtual environment for the autonomous system, rather than the input signals. The input signals are generated in the virtual environment during simulation based on the actions of the autonomous agent.

Search-based algorithms are a common choice for test case generation, given their ability to effectively explore large search spaces and produce diverse test scenarios [72]. Moreover, they are customizable, black-box optimization algorithms that do not require access to objective function gradients [40]. However, existing search-algorithms are used for general purpose optimization and are not tailored for autonomous RS testing [73]. It is thus critical to leverage the operational knowledge of the system to guide the test generation. A number of approaches have been proposed to embed such operational knowledge into the search process, which we review below.

3.1 Operational Knowledge in Search-Based Testing

Our main observation from test generation approaches is that, while operational knowledge can be beneficial, it may also introduce bias and, in some cases, focus the search on regions of the input space that are unlikely to reveal failures. In this thesis, we aim to develop approaches that enable more effective discovery of failure-inducing scenarios while minimizing the bias introduced by operational knowledge.

3.1.1 Domain-Specific Objective Functions and Search Operators

In search-based autonomous RS testing, operational knowledge is often expressed in the form of objective functions used to guide the search [74]. These functions are commonly synthesized from the systems requirements, and running full simulations is needed to evaluate them [75]. Luo et al. consider traffic scenarios, with road intersections and other vehicles [76]. They leverage a multi-objective genetic algorithm, NSGA-III, for finding test scenarios violating different combinations of the seven established system requirements. Abdessalem et al. [35] consider testing an AV system with multiple features, as Autonomous Cruise Control (ACC), Traffic Sign Recognition (TSR), Pedestrian Protection (PP), and Automated Emergency Braking (AEB) in traffic scenarios with road signs, pedestrians and other vehicles. They propose FITTEST approach to detect feature interaction failures by generating test cases with evolutionary search. The authors define a set of hybrid test objectives that combine interaction coverage metrics with metrics describing safety requirements violation. In DeepHyperion [77] diversity based objectives are introduced, defining an n-dimensional feature grid constructed from operational-knowledge-driven characteristics. The search is then guided toward maximizing the number of grid cells covered by failure-inducing test scenarios. Tuncali et al. [78] aims to maximize the discrete parameter combination coverage in emergency braking system testing, while CRAG [79] maximizes the number of high-level road configurations covered for autonomous lane keeping testing. Despite coverage-oriented objectives, the search space remains large, and exploration effectiveness is limited by the testing budget, as failure identification relies on requirement-based objective functions that need full simulation execution.

To reduce reliance on computationally expensive simulations, in AmbieGen tool [80] operational knowledge in the form of vehicle kinematic models is used to guide the search. Formica [73] et al. propose ATheNA framework that combines both, manually defined fitness function based on engineers' domain knowledge and the fitness functions generated automatically from system requirements specifications. The authors aim to combine the benefits of inexpensive fitness function approximations as well as simulation based evaluations.

Search operators play an in important role in the search space exploration [40]. Thus, a number of approaches develop customized search operators alongside requirement-based fitness functions. As one of the first EA-based test scenario generation tools, Gambi et al. [38] proposed AsFault. AsFault leverages a single-objective genetic algorithm to generate challenging road topologies to test an autonomous vehicle lane-keeping assist system, where the objective to guide the AV towards going out of the road lane. Mutation and crossover operators operating at the road segment level are used. A similar test subject was used by a number

of other search-based approaches, such as DeepJanus [81], Frenetic [82] and OptAgle [83]. In DeepJanus, a two-objective genetic algorithm NSGA-II is used to guide the system towards the frontier of correct and incorrect behaviors, maximizing the difference between the failure scenarios that were found. Frenetic employs a single-objective genetic algorithm with custom mutation and crossover operators for road topology manipulation, along with tailored operator selection rules. OptAngle proposes an angle-based road spline representation and corresponding search operators.

Another group of approaches focuses on obstacle placement use cases for testing autonomous drones [84], [85]. CAMBA [86] relies on rule-based geometric mutations to specify the positions of two obstacles in the environment and make the drone crash. PALM [87] leverages Monte Carlo Tree Search and mutation operators for obstacle placement, making it hard for the drone to complete the mission safely. OptObstacles [88] leverages multi-objective meta-heuristic search, positioning obstacles close the expected trajectory of the drone.

In the next group of approaches, the authors consider more complex test scenarios for autonomous vehicles, including driving in traffic with multiple other vehicles, pedestrians, intersections, and traffic signs. Recently, Huai et al. [89] have proposed a DoppelTest test scenario generation approach that utilizes a genetic algorithm to discover bug-revealing violations in driving scenarios involving multiple vehicles, pedestrians and defined traffic control signals. In test scenarios, vehicles and pedestrians are represented by the waypoints of their trajectories as well as their speed. DoppelTest leverages NSGA-II [90] algorithm guided by 4 objectives, encouraging the AV instances to be in close proximity to other road traffic participants to increase the likelihood of their interaction. Custom search operators such as adding or removing traffic participants are used. Huai et al. [91] develop the SCENORITA approach, whose main contribution is the introduction of fully mutable obstacles, referring to traffic participants such as pedestrians and other vehicles. The proposed mutations include modification of obstacle attributes, type, or quantity.

DriveFuzz [92] applies fuzzing-based testing to autonomous driving systems by introducing a set of domain-specific mutation operators, including road map modifications, actor behavior perturbations, puddle placement, and weather variations. The search process is guided by simulator-evaluated objectives, such as collision occurrence, traffic infractions, and vehicle immobility. Li et al. [93] propose to combine genetic algorithms with local fuzzers to test AVs in a traffic environment. The objective is to efficiently identify traffic maneuvers of surrounding vehicles that lead to AV safety violations. The authors use genetic algorithms to guide the search towards problematic situations where the AV's safety is low with objective function defined based on domain knowledge of safety rules. They then employ a local fuzzer

to extensively explore each of these high-potential cases and evolve them into safety-violating scenarios by mutating traffic participants maneuvers. While custom search operators help in improving the search effectiveness, such approaches tend to remain specific to particular use cases or even individual system modules under test, thus the cost of applying them to a novel system under test remains high, as the operators need to be redesigned. Contrary to these approaches, we advocate leveraging available operational knowledge to shape the search space.

Conclusion. As observed, many of the reviewed approaches rely on test-problem-specific operators to modify test scenarios, as well as requirement-based objective functions to guide the search. However, the underlying search space remains vast, and the effectiveness of these approaches is constrained by the available evaluation budget. The proposed search operators are typically tailored to specific systems under test and require manual design effort. In the next subsection, we focus on search approaches that emphasize efficient exploration of the search space by learning the operational knowledge from the system evaluations, reducing the manual effort required to design specialized search operators and fitness functions.

3.1.2 Learning Operational Knowledge from Data

This category of approaches introduces learning components that leverage data from previous executions to learn and predict failure related events in test scenarios, thereby reducing reliance on full simulation-based evaluations.

One of the earliest works to incorporate machine learning with ES in the context of test scenario generation is the study by Abdessalem et al., [94] who develop a multi-objective genetic algorithm based on NSGA-II for testing autonomous vehicle pedestrian detection system. In order to reduce the execution time of the search algorithm, the authors propose a new combination of multi-objective search with surrogate models built based on neural networks [18]. Surrogate models are able to predict the fitness function values within some confidence intervals. The time required for surrogate models to predict values is significantly less than the time required to run simulations of physical models. The idea of the approach is to bypass the execution of the simulation for any individual, whose predicted fitness value is lower, than a certain threshold. One limitation of this approach is the use of supervised models, which may perform poorly when dealing with inputs that have distributions that differ from the training dataset. In their future work, Abdessalem et al. [39] used decision trees (DTs) as classification models that help focus the search on the critical test input spaces for efficient online testing of vision-based advanced driver systems (ADS). The test scenarios included various weather conditions, road topologies, and locations of other vehicles and

pedestrians. DTs were used to classify the quality of a candidate solution. However, the approach has a limitation in terms of scalability, particularly with higher dimensions. For instance, when the input is high-dimensional, DTs would require a significant number of examples and iterations to learn a meaningful representation. More recently, Sorokin and Kerscher [95] proposed using support vector machines (SVMs) instead of decision trees for fitness prediction in this context, aiming to obtain more accurate decision boundaries.

Birchler et al. [96] use ML to identify simulation-based tests that are unlikely to detect faults in the autonomous vehicle software under test and skip them before their execution. They use three types of basic ML models: logistic regression, Naive Bayes and Random Forest. The F1 of the reported models is between 47% and 90%, indicating that the tool may mistakenly discard some fault-revealing test cases. Moreover, creating the datasets is time-consuming and computationally expensive, requiring a big number of evaluations of a simulator-based model. Haq et al. develop SAMOTA (Surrogate-Assisted Many-Objective Testing Approach) [24], an approach for test suite generation that utilizes multiple surrogate models, one for each search objective, to perform the explorative global search. Then local surrogate models are created, one for each failure cluster, exploiting the promising search space regions detected in the global search phase. DeepAtash-LR [97] introduces a surrogate model into the illumination-based search originally proposed in DeepHyperion [77]. The surrogate model is trained on datasets that record system execution traces in simulation along with their corresponding fitness values. Training is performed once a sufficient number of inputs has been accumulated during the search process. During the search, model uncertainty is taken into account and estimated as the prediction deviation across an ensemble of surrogate models. Zolfagharian et al. [98] proposed STARLA, a search-based approach for testing Deep RL agents. To reduce the computational overhead of running the RL agent in a simulator, the authors employed a Random Forest ML model to predict episode outcomes and assign approximate fitness values to the test scenarios (test episodes for the RL agent).

Peltomäki et al. [99] propose the WOGAN approach, which utilizes a WGAN [100] architecture to learn the distribution of tests that align with the distribution of safety-critical tests for autonomous systems. Tests are sampled from the learnt distribution and prioritized based on the 'analyzer' module predictions. Winsten et al. [101] apply WOGAN for test generation in autonomous UAVs. To prevent the generation of invalid test scenarios when training the WGAN generative model, the authors impose several complex geometrical constraints on the test representation. Recently, Peltomäki et al. [102] introduced a surrogate model to predict the outcomes of tests generated by a GAN-based approach. The surrogate model is trained online and does not require pre-training.

Another category of approaches for generating test scenarios is based on active sampling methods. These methods allow the learning algorithm to have some control over which data points to select for learning, rather than relying solely on the provided data. Cross-entropy sampling [103] is one example of such active sampling methods. O'Kelly et al. [104] use a cross-entropy sampling method to generate challenging driving scenarios involving up to six driving participants. Dreossi et al. [27] have developed a software toolkit, VERIFAI, for the formal design and analysis of cyber-physical systems that include AI and ML components. To specify test scenarios, users can utilize SCENIC [105], a probabilistic programming language for modeling environments, combined with a renderer or simulator for generating sensor data. To guide test generation, VERIFAI uses active samplers, such as cross-entropy and Bayesian optimization. Bayesian optimization is a technique for finding the optimal set of parameters for a given objective function by iteratively updating a prior probability distribution with information from the evaluated points and selecting new points to evaluate based on an acquisition function that balances exploration and exploitation [106]. Based on VERIFAI, Ramakrishna et al. [107] propose ANTI-CARLA, an automated testing framework in CARLA simulator [108] for simulating adversarial weather conditions (e.g., heavy rain) and sensor faults (e.g., camera occlusion) that fail the system. To guide scene generation, the authors use two samplers: Random Neighborhood Search (RNS) and Guided Bayesian Optimization (GBO). These samplers extend the conventional random search and Bayesian optimization search methods [109]. Mullins et al. [110] employ adaptive sampling to intelligently explore the state space and identify test scenarios that lie on the boundary between distinct performance modes. They also use unsupervised clustering techniques to group scenarios according to their performance modes and rank them based on their effectiveness in diagnosing changes in the autonomous system's behavior. To achieve this, they utilize Gaussian Process Regression [111], which is a type of Bayesian optimization, to model the autonomous system's performance and selectively target regions that might indicate performance boundaries. Their study focuses on AV testing with simple scenarios, where the test vehicle must navigate towards a goal location while avoiding static square obstacles.

Conclusion. Overall, we can see that operational knowledge can be learnt from simulator-based executions and used as an inexpensive prediction in the future search process, improving search efficiency. At the same time, these models are only limited to the data obtained from the simulated environments, rather than the real world. Moreover, they serve only as the approximation of the system behavior in simulation and may still wrongly direct the search. The surrogate models depend on the data used to trained them and do not generalize beyond the data seen during the training. Some approaches explicitly account for model uncertainty when encountering novel inputs and use this information to guide the search, par-

tially mitigating generalization limitations. We therefore consider the integration of surrogate models with uncertainty quantification to be one of the promising directions for future work. Currently, this research direction remains orthogonal and potentially complementary to the approaches proposed in this thesis. Surrogate models with uncertainty quantification can be leveraged in the form of operational knowledge in the proposed approaches, particularly in RIGAA and RILaST.

3.1.3 Search Initialization

A number of previous works in software testing domain confirmed the importance of having good seeds in the initial population and developed methods to generate these seeds. Moghadam et al. [112] utilize an initial quality population seed to enhance the search process. The quality population consists of a combination of valid random solutions and solutions that achieved high fitness in earlier runs of the tool, which can be considered as previously acquired system operational knowledge. Fraser and Arcuri [113] proposed several seed generation strategies for the whole test suite generation problem (each individual corresponds to a test suite). One of them is a tournament strategy where only the best solution (in terms of line or branch coverage) out of N randomly produced solutions is selected for inclusion in the initial population. Lopez et al. [114] suggested a 'single-objective based seeding strategy' to produce seeds for the whole test suite search. The idea is to first generate a complete test suite (one individual) with a single-objective genetic algorithm and then use it as a seed for the search algorithm. However, in this case it will be necessary to perform multiple evaluations or run the search to produce each seed, which may become a computational overhead. Arrieta et al. focus on improving the diversity of the initial population [115]. They propose an Adaptive Random Population Generation (ARPG) algorithm, where the individuals to be added to the initial population are selected based on their distance from the already included individuals. N random seeds need to be produced and only one that is the farthest from the existing individuals is selected. This approach is inspired by the Adaptive Random Testing (ART) technique proposed by Chen et al. [116], which prioritizes the test cases to be executed. In ART, the next test is selected based on its distance from already executed tests.

Other approaches leverage operational knowledge in the form of real-world failures as seeds to direct the search. Gambi et al. propose CRISE [117], that creates accurate car crash simulations from accident sketches. SURREALIST by Khatiri et al. [118] analyzes logs from real UAV flights and automatically generates simulation-based test cases in the neighborhood of these flights by applying custom mutation operators. The STRETCH tool by Scheuer et

al. [119] extends recorded real-world accidents to generate avoidable test scenarios, that is, scenarios in which a correctly operating vehicle under test could avoid the collision. This is achieved using a multi-objective optimization algorithm.

Conclusion. Overall, we can see that existing approaches bias the initial population toward more diverse or previously known failure regions. A key limitation of diversity-guided initial sampling is that only a small fraction of the generated seeds may actually lead to failures. Similarly, when existing failures are used as seeds, the search tends to focus on exploring regions in the vicinity of known cases rather than discovering entirely new failure modes. In contrast, in our work we aim not to limit the search to already known tests, but rather incentivize it to discover novel failure-inducing scenarios. In the RIGAA approach, we leverage a reinforcement learning agent that discovers new failures through a learning process guided by operational-knowledge-based reward functions. The newly discovered failure patterns are then used as seeds to initialize the subsequent search.

3.1.4 Search Space Representation Improvement

Another important part of the search process is the input space representation [40]. In the CPS testing literature, research on representation improvement remains limited. A number of works focused on the problem of representing road topologies for the purpose of autonomous LKAS system testing. Gambi et al. [38] in their AsFault approach propose building a graph representation of the road network, in which edges model road segments and nodes model either road intersections (internal nodes) or intersections between roads and map boundaries. This representation is better suited for evolving road networks than working with polylines directly. Castellano et al. [120] empirically compared six different representations allowing to map road representation in 2D space to a 1D space of values. The results showed that the representation based on a sequence of curvature values of Frenet frames [121], referred to as 'kappa' representation, outperforms the other alternatives based on Catmull Rom splines and Bezier curves. In EvoScenario, the authors go one step further and propose to encode the road segments as well as the movement of dynamic objects, such as adversarial vehicles [122].

Other group of approaches is focused on the generation of traffic scenarios. Lu et al. [123] propose an approach with adaptive representation named EpiTESTER. In EpiTESTER, the test scenario representation is evolving along with the test parameters, making some parameters more important in the search than the others. This approach is based on the epiGA [124] algorithm, which keeps a population of chromosomes and corresponding nucleosomes, that indicate what genes can be mutated. The intuition is that during testing, as the environment changes, some scenario parameters become more important than others. Huai et al. [91] de-

velop the SCENORITA approach, where the main contribution is in introducing the so called "fully mutable obstacles". These are obstacles whose parameters can be modified not only during the mutation stage, but also during the crossover. This is achieved by representing obstacles as individuals instead of genes.

As observed, existing approaches for ARS testing primarily focus on problem-adapted representations defined in the original input space, often derived from the knowledge of the test generation problem. However, these approaches do not exploit operational knowledge in the form of evaluation functions to improve the input representation and, consequently, the structure of the search space. At the same time, optimizing the search space was shown to be effective in the evolutionary search literature.

Gaier et al. [125] proposed Data-Driven Encoding MAP-Elites (DDE Elites), which is one of the first works to augment the evolutionary search with the search in latent space. The authors describe an approach for generating a dataset of high-performing and at the same time diverse solutions. They leverage MAP-Elites algorithm [126] in combination with the multi-armed bandit strategy to manage the exploration-exploitation trade-off. They show the effectiveness of their approach on the control problem of a 2D planar arm with 20, 200, and 1000 degrees of freedom (Dof). In their experiments, the dimensionality of the latent space is 10, 32, and 200 corresponding to each Dof. Bentley et al. [127] propose Constrained Optimization in Latent Space (COIL) approach, where variational autoencoder (VAE) is used to learn a new representation for search problems that is biased towards solutions that satisfy the problem constraints. Unlike DDE-Elites approach, the learned representations do not involve a reduction of the number of parameters; instead, the main goal of COIL is to remap a hard-to-search genotype space into a more focused easier-to-search latent space. To generate the dataset for VAE training, the authors relied on multiple runs of GA, where constraint satisfaction degree was used as an objective function. The search in the latent space is performed with GA with a simplified configuration, since the latent space is optimized. However, in all the conducted experiments, the authors used only a simple objective function in the form of the squared sum of variables with two different constraints. In their recent work [128] Bentley et al. evaluate COIL on the real world optimization problem of configurations of autonomous robots for last-mile delivery. COIL was able to find valid solutions for all problem variants, outperforming standard GA. The authors further extended their work and proposed the SOLVE: Search space Optimization with Latent Variable Evolution approach [129]. In SOLVE, the main focus is to bias the representations toward specific criteria or heuristics, including, but not limited to, the satisfaction of constraints. However, SOLVE does not guarantee constraint satisfaction, as its primary goal is search space optimization. SOLVE has only been evaluated using a simple objective function with 5 distinct

constraints. Unlike SOLVE, the proposed RILaST approach focuses not only on optimizing the search space, but also on embedding operational knowledge into the definition of the search space. Moreover, it ensures the generation of diverse, failure-revealing test scenarios in settings where the objective function relies on computationally expensive simulator-based evaluations.

Conclusion. Overall, existing approaches for testing autonomous robotic systems leverage problem-specific knowledge to enhance representations in the original input space. Inspired by recent advancements in evolutionary search, we advocate for improving representations in the latent search space. This enables the use of operational knowledge–based objective functions to shape the space effectively. Simulation-based evaluations are then performed within this improved latent space to accurately estimate requirement-driven objective function values, thereby increasing the efficiency of the search and avoiding invalid predictions from the approximated objective functions.

3.1.5 Interactive Adversarial Behavior Generation

Previously we reviewed approaches for testing autonomous robotic systems that focus mostly on the manipulation of static environment elements, such as the positions of objects or obstacles, target path geometries, and initial conditions of scenarios involving multiple traffic participants, or combinations thereof. These scenarios differ from interactive scenarios involving dynamic elements, in which adversarial agents can change and adapt their behavior during scenario execution through continuous interaction with the system under test [36].

Reinforcement learning is most commonly used to generate interactive adversarial behaviors [30]. At the same time, RL algorithms are not inherently designed for test scenario generation [48]. Consequently, reward functions, as well as exploration and exploitation strategies, should be adopted to the given autonomous RS testing tasks.

One of the earliest works to apply RL to testing autonomous RS is adaptive stress testing (AST) framework by Lee et al. [130]. AST aims to find the most likely path from a start state to a failure state in a discrete-time simulator. AST formulates the search as a reinforcement learning problem by considering a reinforcement learning agent A that acts as an adversary to the system under test. The agent chooses disturbances so that the environment is as challenging to the system under test as possible. Lee et al. [131] use AST for aircraft collision avoidance application testing in a scenario with two aircraft. They guide the search based on a near midair collision metric, that estimates the minimal distance between aircraft to be maintained. Delecki et al. [132] applied adversarial stress testing to evaluate the robustness of LiDAR-based autonomous vehicle perception systems under adverse weather conditions.

They simulated LiDAR point clouds with disturbances caused by heavy rainfall rates of 20, 30, and 40 mm/h, and trained an RL agent to select the best rainfall rate at each simulation time step. However, the study is limited by the simplicity of the scenario and the small action space of the RL agent, which significantly simplifies the task of training it. As we see, AST approaches, are more suited for generating dynamic disturbances, rather than complex adversarial behaviors.

We further discuss more adversarial behavior oriented approaches. Karunakaran et al. [133] use deep reinforcement learning to generate pedestrian behaviors falsifying an AV advanced braking system. The agent is rewarded for taking actions that lead to failure as measured by the safe distance. This approach considers a simple scenario with only two agents (a pedestrian and a car) and has a low-dimensional search space. Chen et al. [134] propose an adaptive evaluation framework to evaluate autonomous vehicles in adversarial lane-change scenarios generated by deep reinforcement learning. The idea is to train the vehicle under test as well as the other vehicle on the road i.e. adversary vehicle at the same time. The vehicle under test is rewarded for safe and correct driving behavior, whereas the adversarial vehicle receives a reward that is directly inverse to that of the system under test, thereby incentivizing the discovery of misbehaviors in the SUT. MORLOT [36] adopts RL test generation to ensure the coverage of specified requirements while conducting dynamic testing. The test scenarios specify the actions of pedestrians, other vehicles, as well as weather conditions. The authors propose to have a separate Q-table for each objective during the RL agent training, that can be re-used in future runs. This approach employs one central RL agent controlling all the environmental parameters. DeepCollision [29] approach leverage Deep Q-learing to select environmental parameters in the dynamic scenarios for pedestrian collision avoidance. Scenarios involve selecting the SUT and pedestrian position, weather and time of the day. The notions of safe distance [135] and collision probability are used to calculate the RL agent reward. At the same time, this approach focuses more on dynamic parameter specification, rather than generation of reactive behaviors. More recently, Doreste et al. [30] proposed modeling each traffic participant as a separate reinforcement learning agent for scenario generation. Their work considers a highway environment with two vehicles, where an adversarial reward function encourages failure events while penalizing overly aggressive behavior. However, the approach does not distinguish between valid (i.e., avoidable) and invalid failures during reward computation.

One of the main limitations of the described approaches is the lack of explicit incorporation of operational knowledge in the form of physical and safety constraints to ensure the validity and realism of the generated scenarios. Moreover, these approaches primarily focus on dynamic behaviors, without explicitly optimizing static environmental elements, such as initial

conditions. A limited number of works explicitly incorporate scenario validity rules [93, 136]. However, these approaches are predominantly fuzzing-based and do not involve dynamic decision making. In such methods, validity rules are typically enforced as search constraints rather than being treated as explicit optimization objectives.

Conclusion.In existing approaches to dynamic scenario generation, operational knowledge is most commonly encoded as rewards for reinforcement learning agents. However, the approaches rarely incorporate explicit scenario validity constraints into adversarial reward shaping, which can lead to unrealistic or overly aggressive behaviors. Moreover, existing approaches do not systematically explore variations in static initial conditions, which also play a role in dynamic scenarios. The proposed DYNASTO approach separates dynamic and static optimization. In Step 1, a validity-aware adversarial policy is learnt with RL, guided by temporal-logic constraints. In Step 2, the genetic algorithm searches over initial conditions while replaying the learned adversarial behavior, systematically exposing failures that emerge from initialization.

3.2 Operational Knowledge for Sim2real Gap Minimization and Real World System Improvement

Simulation-based testing is an important stage in the development of autonomous robotic systems. Its effectiveness, however, largely depends on the similarity between the real world and the simulated environment. This challenge, known as the simulation-to-reality (sim-to-real) gap, must be carefully considered during in-simulation testing and development [137].

Stocco et al. [137] were among the first to empirically demonstrate discrepancies between real-world autonomous driving systems and their virtual counterparts. Their study highlights that the simulation-to-reality gap is largely due to limited photorealism and simplified modeling of vehicle sensors and the surrounding environment. To mitigate these discrepancies, the authors propose the use of multiple redundant digital twins with complementary strengths, allowing the weaknesses of individual simulators to be compensated within a multi-simulator testing setup. n response, Biagiola [64] proposed the "Two Is Better Than One" approach, in which two different simulators are used to test the same lane-keeping assist system. Their results indicate that these digital siblings are effective at predicting the failures of the autonomous vehicle under test when compared to a high-fidelity digital twin of a physical scaled vehicle performing the lane-keeping task. Sorokin et al. [138] propose MultiSim, an approach that couples evolutionary search with multi-simulator execution. During the search process, each test scenario is evaluated across three different simulators, and scenarios that exhibit consistent behavior and outcomes are prioritized for further exploration. While

leveraging multiple digital siblings is a promising direction, it leads to a linear increase in both test setup complexity and execution cost. Consequently, we advocate for the use of a single, highly realistic simulation environment.

As mentioned previously, photorealism plays a very important role in reducing the perceived simulation to reality gap. Unpaired style translation generative models, such as CycleGAN [59] and its variants, have been widely used to transform simulator renderings into more realistic styles [28, 64]. More recent work has explored diffusion models for domain augmentation, expanding visual conditions and uncovering additional failures before on-road testing [139]. However, these techniques do not explicitly guide test generation toward failure-inducing corner cases.

Another group of approaches focuses on reducing the sim-to-real gap at the perception system level, without considering closed-loop system execution. Several works study the differences between online evaluation, where perception components are assessed as part of a full closed-loop system simulation, and offline evaluation, where only the perception module is tested in isolation [28, 140]. Stocco et al., [28] extend the study by Haq et al., [140] with a physical set-up and conclude that deviations between off-line and on-line discovered failures can be minimized with better scenario matching techniques. In our work, we perform both, on-line DNN testing in MARTENS and off-line DNN testing in STRIM.

Several techniques have been proposed for repairing DNNs using intelligent data augmentation methods, such as style transfer in DeepRepair [141], search-based data augmentation in SENSEI [142], and image-to-image translation of different environmental conditions in TACTIC [143]. In DeepRepair, the authors leverage generative style transfer models to extract realistic noise patterns from real-world data and combine them with noise characteristics specified by domain experts. These models are then used for data augmentation and DNN training. SENSEI on the other hand leverages only genetic search for perturbation selection as does not rely on generative models. TACTIC leverages multimodal image-to-image translation [144] to learn styles of failures inducing input images (often attributed to specific weather conditions) and applies them to other generated images.

Fahmy et al. [34] propose SEDE, a DNN testing technique that combines root-cause explanation with real-world system fine-tuning. The approach relies on genetic algorithms to steer the simulator toward generating images that resemble failure-inducing real-world samples from a test set. It then employs rule-learning algorithms to derive expressions that describe these failures. The extracted expressions are subsequently used to generate additional synthetic images, which are leveraged to retrain and improve the DNN. The approach is evaluated on automotive perception tasks, such as gaze direction estimation. These approaches consider

only offline DNN testing and are not adopted to autonomous RS testing.

Attaoui et al. [145] developed the DESIGNATE approach, using paired style transfer generative models to transform simulator-generated data into more realistic images for DNN retraining. This model is combined with a search algorithm to reveal DNN mispredictions. The discovered failure-revealing inputs are used for the original DNN repair. Similarly to our work, failure revealing inputs are used to fine tune the original DNN model. Unlike DESIGNATE, in our work the real-world and virtual environments are closely aligned, as the simulated environment is reconstructed using assets from its real-world counterpart. MARTENS reduces the sim-to-real gap by leveraging labeled real-world data, whereas STRIM utilizes unlabeled data in combination with a style transfer model. Notably, STRIM employs unpaired style transfer methods, which are more practical in real-world scenarios where paired images are generally unavailable.

Amini et al. [31] propose a novel unpaired generative style transfer model based on variational autoencoders, tailored to the autonomous driving testing domain. They apply the proposed style transfer model in the on-line DNN testing setting. The authors demonstrate that employing style translators can reduce the need for online, closed-loop DNN testing as the style translator model increases the correlation between offline and online testing. However, the authors do not perform a guided search towards failure revealing inputs. Moreover, they emphasize the need for labeled real-world datasets for training DNN models, whereas the STRIM approach seeks to minimize manual data labeling by training and fine-tuning the DNNs on style-transferred images.

Conclusion. The simulation-to-reality gap is a well-recognized challenge in the search-based testing community. To mitigate this issue, operational knowledge is often incorporated in the form of multiple test oracles (e.g., simulators), prior knowledge of failure-prone inputs, or real-world data. However, many existing approaches primarily target general computer vision problems, while autonomous robotic RS–specific use cases remain comparatively underexplored. Moreover, a large body of work relies on existing datasets, which may differ substantially from both the simulation-based setup and the envisioned operational conditions of the system under test. We argue that virtual environments should approximate real-world conditions as closely as possible in order to provide more relevant operational knowledge. In MARTENS, we therefore rely on highly photorealistic simulation environments built from 3D assets that closely match their real-world counterparts. In STRIM, we take this a step further by performing direct 3D scanning of the target deployment environment.

3.3 Chapter Summary

From the analyzed works, it is evident that operational knowledge plays a crucial role in autonomous system testing. The reviewed literature demonstrates problem-specific search operators and requirement-driven objective functions derived from operational knowledge are very common in search-based testing. While these approaches improve effectiveness over naive sampling, they remain constrained by vast search spaces, limited evaluation budgets, and the substantial manual effort required to design system-specific operators and fitness functions. To address these limitations, recent works have explored learning operational knowledge directly from simulator-based executions, for example through surrogate models, to guide search more efficiently. However, learned models are inherently bounded by the data used for training, may fail to generalize to unseen regions of the search space, and can misdirect the search when faced with novel inputs. Approaches incorporating uncertainty quantification partially mitigate these issues and represent a promising, complementary research direction that can be leveraged as operational knowledge within frameworks such as RIGAA and RILaST. Existing methods also direct the search toward known failure regions through reuse of prior failures, which often limits the discovery of fundamentally new failure modes. In contrast, our work emphasizes incentivizing the discovery of novel failures by learning from operational knowledge guided interactions, notably through RL-based exploration in RIGAA and representation improvement in latent search spaces in RILaST. Furthermore, in dynamic scenario generation, operational knowledge is commonly encoded solely via reward shaping, often without explicit validity constraints or systematic exploration of static initial conditions. The proposed Dynasto approach addresses these gaps by decoupling dynamic behavior learning from static parameter optimization and enabling validity-aware adversarial behavior learning. Finally, we observe that many of the existing approaches for simulation to reality gap reduction are for general purpose perception models, rather than addressing the use cases more common in autonomous RS domain. The works that focus on robotic systems, are mostly autonomous driving centered. Moreover, these approaches are oftentimes relying on generic datasets or simplified simulations. Our MARTENS and STRIM are designed for autonomous robotic manipulator testing, which is a novel direction in the field. In these approaches, we ensure that simulation-based testing setup remains closely aligned with real deployment conditions.

CHAPTER 4 REINFORCEMENT LEARNING INFORMED EVOLUTIONARY SEARCH FOR AUTONOMOUS SYSTEMS TESTING

4.1 Problem Context

Thorough testing of autonomous robotic systems, such as self-driving cars, autonomous robots, and drones is a crucial step in their development cycle. An important testing stage is software-in-the-loop or model-in-the-loop, where the system model is run in a virtual environment [146]. Test scenarios are typically represented by virtual environments, where the system model should accomplish a particular task. However, the search space of possible parameters defining the test scenarios is huge, and simulating all the parameter combinations is computationally infeasible. It is important to have search-based techniques that generate scenarios with an increased probability of revealing a system failure. These scenarios can then be prioritized for testing the system.

Recently, a number of search-based techniques were successfully applied to test case generation for autonomous systems. In the context of testing, a good test scenario is the one that makes the system falsify some of its requirements, i.e., reveals its failure. Gambi et al. [38], Riccio and Tonella [147], Castellano et al. [82], Zohdinasab et al. [77] generate test cases for vehicle lane keeping assist system (LKAS) with evolutionary search algorithms (EA). Abdessalem et al. [39], [35] use a multi-objective search evolutionary algorithm NSGA-II to obtain test scenarios for autonomous vehicle automated emergency braking (AEB) system. One of the main limitations of these approaches is the use of the computationally expensive simulator-based system model to perform the fitness function evaluations. A simulator often takes a non-negligible amount of time to evaluate a candidate solution, ranging from tens of seconds to minutes. Moreover, the computational requirements and costs to evaluate each scenario are high. As an example, the minimal requirements for running the BeamNG simulator [148] are 16 GB RAM, Nvidia GeForce GTX 970 videocard and Intel Core i7-6700 3.4Ghz processor or better. Given that each evaluation is expensive, we are interested in approaches that allow to minimize the use of computational power on the evaluation of non-promising solutions, i.e., those that are very unlikely to lead to a system failure. Evolutionary algorithms are effective for exploring large search spaces, however, their execution involves a considerable number of stochastic decisions, such as parent selection, crossover, and mutation [45]. With each evaluation being computationally expensive, they can take a significant amount of time to converge to the promising regions of the search space. This is a disadvantage from the testing perspective, as the testing budget is limited.

One potential solution is to use some heuristics with prior problem knowledge to guide the search [149]. Examples include inserting high-quality solutions into the initial population or using surrogate models to prioritize and select certain solutions [150]. A number of techniques for seeding the search algorithm were proposed for the software engineering related tasks, such as whole test suite generation [151], [152] or software service composition selection [153]. However, many of them are specific to the problems they were designed for i.e. based on parsing of the source code prior to test generation [154], [151] or maximizing the number of different function names called [113]. In the context of autonomous robotic systems testing, seeding techniques often rely on manually generated test scenarios or test scenarios identified by the search process previously [112].

In our approach, we propose to use gradient-based algorithms, specifically reinforcement learning (RL), to automatically and effectively infer the domain knowledge of the problem and then use it to initialize the population of the evolutionary algorithm. RL algorithms have proven to be effective in generating solutions to problems that can be formulated as a Markov Decision Process (MDP) [48]. We have the following intuition: by formulating test scenario generation as an MDP, RL agents can learn the initial constraints of the problem and identify test scenarios with higher fitness compared to randomly generated ones. However, state-of-the-art RL algorithms are known to be sample inefficient, signifying they require a large number of evaluations before being able to learn a meaningful representation [48]. Consequently, training an RL agent using rewards obtained from executing a simulator-based model of the system becomes computationally prohibitive.

In order to further minimize the computational costs, we suggest training the RL agent using surrogate rewards, based on some domain knowledge of what constitutes a challenging test scenario. We surmise that as long as the trained agent can produce solutions of higher fitness than a random generator and of high diversity, it can be useful for evolutionary algorithm initialization. We refer to our approach as RIGAA (Reinforcement learning Informed Genetic Algorithm for Autonomous systems testing). By leveraging RL for initialization, RIGAA is able to converge to better test scenarios within a smaller time budget, than a randomly initialized search algorithm. The RL agent needs to be trained only once and then can be reused for generating the test scenarios. We choose the reward function to be inexpensive to evaluate so that the RL agent training does not impose a significant computational overhead.

We evaluated our approach for test scenario generation on two different systems: an autonomous RL-controlled ant robot navigating in a maze and an autonomous vehicle lane keeping system (LKAS). These two systems have been previously used in other studies, including the SBST 2022 CPS testing competition [155] and RL benchmarking [156]. Results

show that introducing RL-generated individuals with higher fitness levels into the initial population of a genetic algorithm can lead to faster convergence towards more challenging test scenarios. We also demonstrate that using an RL agent trained with surrogate rewards can be valuable for generating test scenarios for a simulator-based system model. RIGAA outperformed NSGA-II by producing 37.4% more failures for the autonomous ant robot in the Mujoco simulator and 34.2 % more failures for the autonomous vehicle in the BeamNG simulator. RIGAA also discovered 15.5 % more failures than CRAG, 23.5 % more failures than AmbieGen tool and 82.1 % more failures than the Frenetic tool, which are state-of-the-art tools for LKAS testing.

In this chapter, we make the following contributions:

1. RIGAA, a search-based tool for testing autonomous robotic systems, allowing to insert some problem knowledge into the search via a pre-trained RL agent;
2. An extensive evaluation of RIGAA, including the assessment of the RL agent's performance and a detailed analysis of the impact of the percentage of RL-produced individuals in the initial population;
3. A publicly available replication package, including the implementation of RIGAA (see Section 4.4.5).

4.2 Problem Formulation

In this section, we formulate the problem of test scenario generation taking into account our representation. We describe the two concrete test scenario generation problems we address in this work. We formulate them as optimization problems, with constraints to produce valid test scenarios and objectives to falsify the requirements of the system under test.

We begin by presenting the definition of the test scenario as well as the representation we use. A test scenario for an autonomous robotic system is a virtual environment in which it operates, as well as the task it needs to accomplish. Similarly to our previous work [157], we represent the virtual environment as a combination of discrete elements E_m, which we refer to as scenario elements. Each scenario element has a fixed number n of attributes A_n that describe its properties. Such representation makes it easy to define and change various parts of a test scenario. The general test scenario representation is shown in Table 8.1. We represent it as nxm matrix, where m defines the number of discrete elements composing the test scenario and n the number of attributes describing each of the elements, i.e. each element E_m is described by attributes A_{1em}, A_{2em}, A_{nem}. While the number of attributes n remains

the same over all the test scenarios for a given use case, the number of elements m can be variable. In practice, we either limit the value of m to some maximal value or keep it fixed. For instance, if our scenario is a road topology, each discrete element E_m may correspond to a segment of the road. The roads can have different numbers of segments. However, too many road segments are likely to result in a road topology that goes out of the map boundaries. Therefore, we limit the maximum number of road segments. Each attribute A_n is used to describe a particular aspect of the road, such as its length, curvature, material, etc. The goal is to find such a combination of scenario elements E_m and the values of attributes A_n, that the virtual environment respects the physical constraints C and forces the system to violate the requirements R.

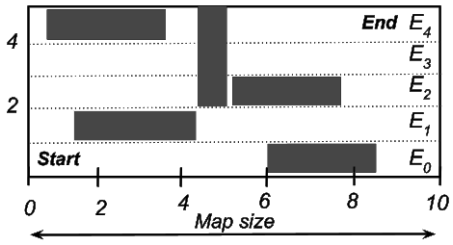
Table 4.1 Test case representation

	E_1	E_2	...	E_m
A_1	A_{1e1}	A_{1e2}	...	A_{1em}
A_2	A_{2e1}	A_{2e2}	...	A_{2em}
...	...	...	...	...
A_n	A_{ne1}	A_{ne2}	...	A_{nem}

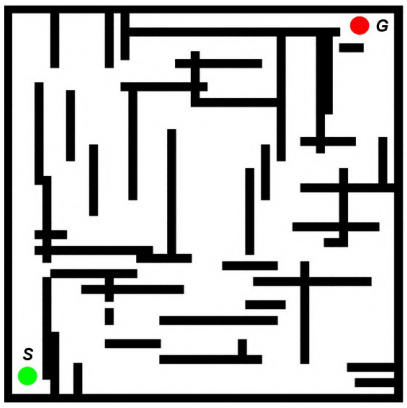
Below, we provide descriptions of test scenario generation problems we considered in our work.

4.2.1 Autonomous Robot Maze

In our study, a test scenario for an autonomous robot constitutes a closed room with obstacles as well as a start and goal location for the robot. Similar test scenarios were used in some previous research works [158], [159]. An example of encoding a simplified scenario is shown in Figure 4.1a and Table 4.2. In this example, the map has a size of 5 m x 10 m, and each scenario element E_m corresponds to an area of 1 m x 10 m, for a total of 5 scenario elements. We specify the type, size, and location of obstacles for each scenario element. An example of a typical test scenario is shown in Figure 4.1b. It depicts a square room of size 40 m x 40 m with vertical and horizontal walls as obstacles. The robot must navigate from the starting location S (green point) to the goal location G (red point) using only its range sensors and planning algorithm. The starting and goal locations remain constant across all scenarios.



(a) Autonomous robot scenario represented in Table 4.2



(b) An example of a test scenario for autonomous robotic system

Figure 4.1 Autonomous robot system scenario examples

Table 4.2 Example of test scenario representation for an autonomous robot

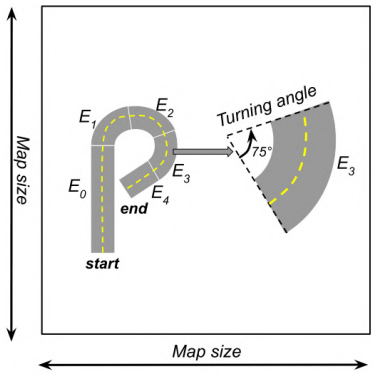
	E_0	E_1	E_2	E_3	E_4
$A_0, wall\ type$	horizontal	horizontal	horizontal	vertical	horizontal
$A_1, wall\ position$	7	3	6.5	4.5	2.5
$A_2, wall\ size$	2	3	2	3	3

In this test scenario, the environment consists of a fixed number m of 40 scenario elements E_m, each representing a 1 m x 40 m area containing one obstacle (horizontal or vertical wall). Each element has three attributes: A_1, the type of the obstacle; A_2, the position of the obstacle (ranging from 2 to 38); and A_3, the size of the obstacle (ranging from 5 to 15). The goal of the test is to evaluate the ability of the autonomous system to navigate from the start to the goal location without getting stuck or colliding with obstacles. Test scenarios should adhere to the following physical constraints C: (1) there must always be at least one path between the start and goal locations; (2) obstacles must not extend beyond the boundaries of the room. The main requirement R for the autonomous system is to reach the goal location safely within the allocated time, without getting into a livelock (wandering around the maze without finding a way out), falling or colliding with obstacles. If the robot does not reach the target location in the defined time budget, the mission is considered failed.

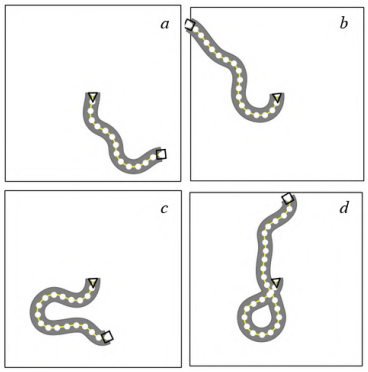
4.2.2 Road Topology for Autonomous Vehicle Testing

In the second problem, the goal is to create a road topology that causes the lane keeping assist system (LKAS) of an autonomous vehicle to fail. The road is represented as a sequence of points on a two-dimensional map with a size of 200 m x 200 m. These points are interpolated using cubic splines to form a smooth road with two lanes. The first and last points on the road define the starting and ending locations, respectively.

An example of encoding a simplified scenario using our representation is shown in Table 4.3 and illustrated in Figure 4.2a. In the example, the road topology is represented as a combination of five scenario elements, each corresponding to a road segment. We define three attributes to describe each segment: A_0, the type of road (straight, right turn, or left turn); A_1, the length of the straight road segment (in the range of 5 to 50); and A_2, the angle of the turn of the curved segment (in the range of 5 to 85). The number of scenarios elements m i.e. road segments is variable, but limited to at maximum 30 elements.



(a) Autonomous vehicle scenario represented in Table 4.3



(b) An example of valid (a) and invalid (b-d) test scenarios for the vehicle LKAS system [160]

Figure 4.2 Autonomous vehicle system scenario example

Table 4.3 Example of test scenario representation for an autonomous vehicle

	E_0	E_1	E_2	E_3	E_4
$A_0, road\ type$	straight	right	right	right	straight
$A_1, length$	15	-	-	-	5
$A_2, turning\ angle$	-	60	60	75	-

The constraints C for the test scenario include: (1) the road has to remain within the map borders, (2) the road cannot be too sharp, and (3) the road cannot self-intersect. We followed

the same constraints that were used at the SBST 2021 tool competition [160]. Examples of valid and invalid scenarios are illustrated in Figure 4.2b, where scenario (a) is valid, scenario (b) features a road that goes outside the map bounds, scenario (c) shows a road with a too sharp turn, and scenario (d) shows a self-intersecting road. The main requirement R for the system is to stay within the lane boundaries, with a threshold of $p = 0.85$ (as defined in at the SBST 2021 competition [160]), meaning that the vehicle is considered to be out of the lane if more than 85% of it lies outside the lane boundaries.

4.3 Approach

In this section, we present RIGAA (Reinforcement learning Informed Genetic Algorithm for Autonomous systems testing), a novel evolutionary search-based approach that incorporates a pre-trained reinforcement learning (RL) agent to generate a percentage of the initial population for generating test scenarios for autonomous robotic systems. Figure 4.3 shows an overview of the approach.

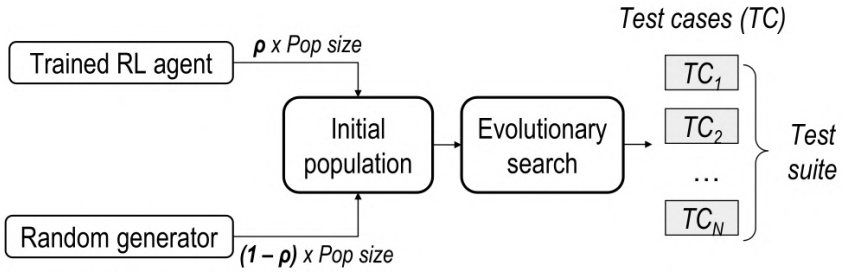


Figure 4.3 An overview of RIGAA approach for test scenario generation

Unlike a traditional evolutionary algorithm (EA), which generates the initial population randomly or using heuristics such as Latin hypercube sampling to ensure a uniform distribution of input solutions [161], we propose using a trained reinforcement learning (RL) agent to generate a certain percentage $\boldsymbol{\rho}$ of the initial population. Unlike random generators that select arbitrary parameters for generating test scenarios, RL enables more informed decisions regarding which parameters to utilize. The motivation to use RL comes from the fact that test scenarios typically contain a number of constraints, and RL agent can learn to produce the scenarios that meet those constraints. Using local search for this purpose, such as greedy-based algorithm, would require additional time to perform the search to produce each solution, while a trained RL agent is capable to produce the scenarios directly. Moreover, if domain knowledge is available, it can be embedded into the reward function, which will allow the RL agent to produce test scenarios taking into account this domain knowledge.

In addition, RL algorithms use a complex gradient-based search strategy [162]. This can allow finding better solutions, than a simple greedy search. RL algorithms also offer a balance between exploration and exploitation. Greedy search is mostly focused on exploitation, while random generator or a diversity guided generation only offers exploration. RL algorithms take both exploration and exploitation into account, therefore, we surmise that RL is a promising approach for test scenario generation.

The RL agent is trained to produce test scenarios with higher fitness than those generated by random generator, while maintaining a high level of diversity. Our hypothesis is that RL-informed initialization will allow the search to find fitter solutions with a smaller evaluation budget, as some previous problem knowledge is already embedded in them. Additionally, it should enable the discovery of new types and patterns of test scenarios. In evolutionary search, individuals with higher fitness have a greater chance of being selected and propagated in the next iterations [45]. Therefore, solutions produced by a trained RL agent have a higher probability of contributing to the algorithm evolution if their initial fitness is higher than that of randomly produced solutions. That is why an important requirement for the RL agent is to produce test scenarios of a higher fitness, than the random generator. Another requirement is to produce diverse test scenarios. After initial population generation, RIGAA follows the typical steps of EA, such as selection, crossover and mutation. At the end of the search, N individuals from the final generation are selected for inclusion in the final test suite. In the following subsections, we provide more details on the implementation of the evolutionary search, the training of the RL agent for test scenario generation, and the overall functioning of RIGAA.

4.3.1 Surrogate Reward

The key novelty of the RIGAA approach is in introducing a pre-trained RL agent for generating some part of the initial population of the test scenarios. The agent is trained to produce diverse and challenging test scenarios. When training the agent we are interested in evaluating to what extent the scenario produced by the RL agent falsifies the system under test. However, as mentioned previously, running evaluations of the simulator-based model is computationally expensive. Each evaluation can take from tens of seconds to minutes. In order for the RL agent to be trained, it needs to interact with the environment for a sufficiently large number of steps. Model-free RL algorithms, such as state-of-the-art PPO algorithm, are sample inefficient [48]. They require a lot of samples (sometimes millions of interactions) for the agent to learn a good policy [163]. In order to reduce the time to perform the evaluations, we suggest using a surrogate reward, based on a simplified approximation

of the system performance. The surrogate reward is a proxy of a true reward function evaluated with the simulator-based system model [24]. Domain knowledge of what constitutes challenging scenarios can be useful in designing the surrogate reward function, however, it is not required. Reward can only be given, for instance, for the discovery of new test scenarios or for producing test scenarios that meet the problem constraints. Below we provide some guidelines in selecting the surrogate function for the RL agent training. We have developed a flowchart that can be used when selecting the surrogate reward function (see Figure 4.4). The surrogate reward can be designed by following the steps of the flowchart. We provide a more detailed description of each step below.

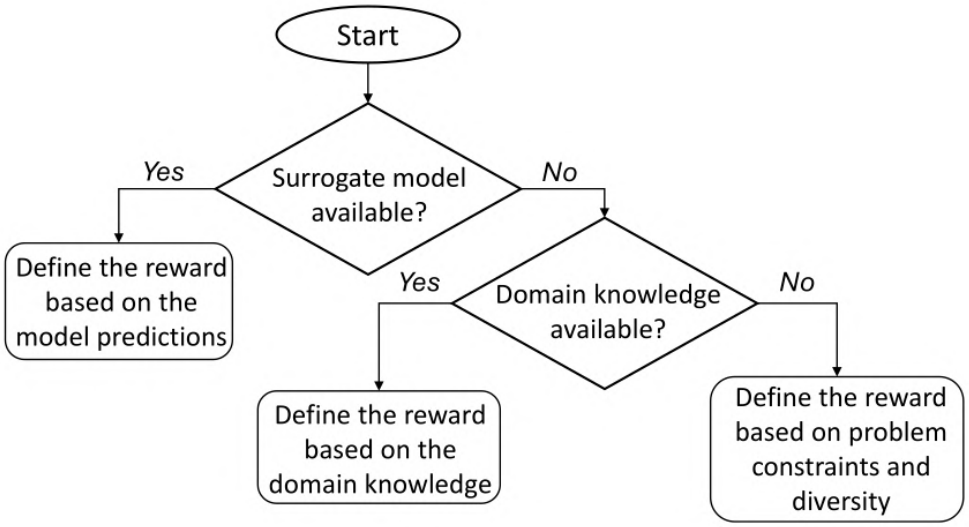


Figure 4.4 A flowchart for selecting surrogate reward function for RL agent training

Surrogate model. By a surrogate model, we assume a model that can produce the approximated outputs for a given system, replacing the expensive simulation-based evaluations. Such models can be derived using traditional system identification methods [71] or supervised machine learning algorithms [96]. For instance, Haq. et al . [24] developed a surrogate model for Pylot [164] system in CARLA simulator. Birchler et al. [96] introduced SDC-scissor, which is a supervised learning-based model predicting the output of the scenarios for LKAS testing. Moreover, some systems, such as mobile robots, have kinematic models available [165], which can also serve to approximate their behavior. The surrogate model's predictions on the test scenario outcome can then be utilized as a reward function during the training of the RL agent.

Domain knowledge. Domain, or operational, knowledge refers to specific characteristics or conditions within a test scenario that present challenges for the system under evaluation.

For instance, road configurations with sharp curves pose a greater risk of failure for LKAS compared to straight roads [77]. By maximizing the curvature of a road topology, we can make it more challenging for the system under test to complete. When the knowledge about such characteristics is not available, we suggest following an open-coding procedure defined by Zohdinasab et al. [77]. This method involves evaluating pre-generated test scenarios with input from human evaluators to identify features that increase scenario complexity. Once scenario characteristics are identified, the RL agent can receive rewards for generating scenarios that possess these characteristics.

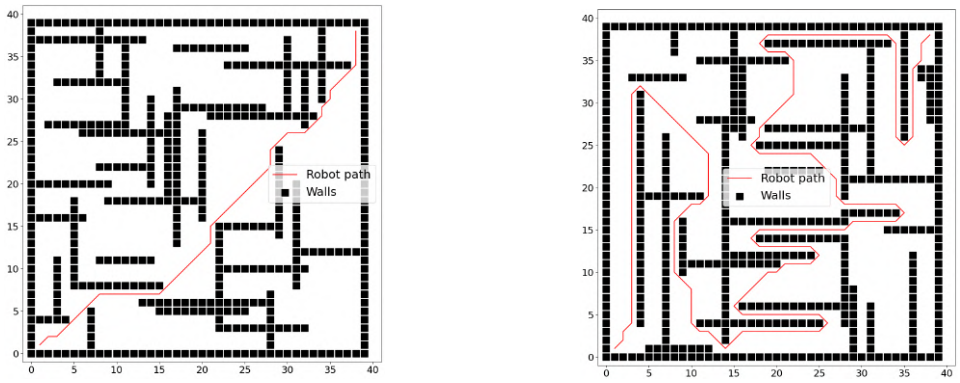
Problem constraints and diversity. If surrogate models are not available and useful scenario characteristics are unknown, it is possible to leverage the test scenario generation problem constraints to provide rewards. The more problem constraints are met by a test scenario, the higher reward can be given to an agent. For instance, when generating road topologies, high reward can only be given for producing valid roads. Furthermore, additional rewards can be earned through the discovery of new scenario elements or their combinations that have not been previously generated. For this, the RL agent will have to keep an archive of all the scenario elements discovered so far. In the following subsections we describe the surrogate rewards we used to train the maze and road topology generating agents.

Surrogate reward function for maze generation

The success of an autonomous robot tasked with navigating to a target location depends on the performance of its planning and control algorithms. Longer paths to the goal location are generally more complex and contain more intricate challenges, requiring the robot's algorithms to adapt and respond to changing conditions. By encouraging the RL agent to optimize the test scenario for longer paths, we can promote the generation of test scenarios that are more likely to expose weaknesses or limitations in the robot's planning and control algorithms. We use the Eq. (4.1) to calculate the surrogate reward for generating maze environments:

$$R_s = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{4.1}$$

where n is the number of trajectory points, (x_i, y_i) are the coordinates of the i-th point, and R_s is the surrogate reward, corresponding to the length of the path in units of distance. An example of scenarios with low and high rewards is shown in Figure 4.5. As you can see, the scenario with the higher reward contains more turns and a longer path to the target, which should present a more challenging environment to navigate.



(a) Example of a test scenario with a lower reward of 58.18

(b) Example of a test scenario with a higher reward of 220.36

Figure 4.5 Example of the vehicle kinematic model trajectory (blue points) given the road topology (yellow points)

Surrogate reward function for road topology generation

Vehicle kinematics is well studied and known [166]. We suggest leveraging some knowledge of the vehicle kinematics as a proxy for the evaluation of the vehicle trajectory on a given road topology. We implement a vehicle kinematic model [165] along with a simplified version of the pure pursuit controller [167] based on the following equations:

$$\begin{aligned}
\delta_{k+1} &= \arctan(\frac{dy}{dx}) - \delta_k \\
\theta_{k+1} &= \theta_k + (\delta_{k+1})\Delta t \\
x_{k+1} &= x_k + v_k \cos(\theta_k)\Delta t \\
y_{k+1} &= y_k + v_k \sin(\theta_k)\Delta t \\
v_{k+1} &= v_k + a_k \Delta t
\end{aligned}$$

where x_k, y_k, θ_k, and v_k represent the current position of the center of the vehicle, orientation, and speed of the vehicle, respectively, at time k. Parameter Δt is the time step between updates, δ_k is the steering angle at time k, a_0 is the acceleration of the vehicle. Values dy and dx are the distances of the vehicle from the y and x coordinates of the closest point on the road lane. In our experiments, we set the model parameters as shown in Table 4.4. We selected the parameters empirically with the goal to maximize the similarity of the trajectories extracted from the simulator-based model logs and the trajectory predicted with the vehicle kinematic model. We evaluated the trajectory similarity based on the Hausdorf distance [168].

Table 4.4 Parameters for the simplified vehicle kinematic model

δ_0	θ_0	ν_0	α_0	Δt
0	0	15	0.1	0.7

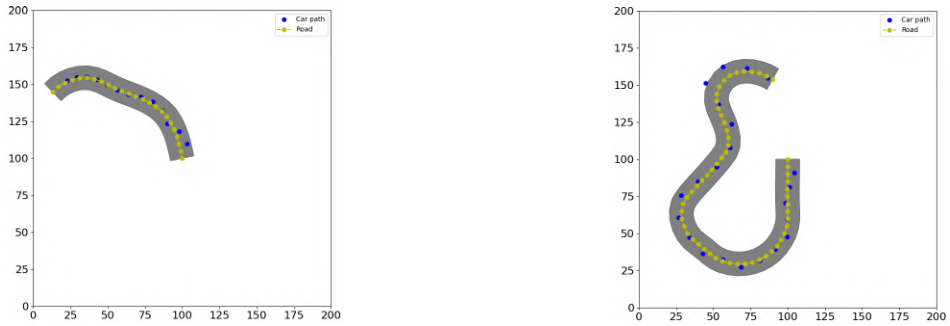
When running the vehicle model with the controller on a given road topology, we record the vehicle location and the corresponding closest point defining the road topology at each timestep k. As a result, we obtain the set of points defining the vehicle model trajectory P_{veh} and a set of corresponding points from the road topology polyline P_{rd}. We evaluate the Euclidean distance between the i-th point in P_{rd} and the i-th point in P_{veh}, corresponding to the vehicle deviation from the road as:

$$d_i = \sqrt{(P_{rd_x,i} - P_{veh_x,i})^2 + (P_{rd_y,i} - P_{veh_y,i})^2}$$

The surrogate reward is calculated as the maximum deviation of the vehicle from the road points:

$$R_s = \max_{i=1}^{n} d_i \tag{4.2}$$

where n represents the number of points in the sets P_{rd} and P_{veh}. Examples of test scenarios with low and high rewards are shown in Figure 4.6. We can see the road topology with a higher reward contains more turns and thus should be more challenging for the driving agent.



(a) Example of a test scenario with a lower reward of 2.13

(b) Example of a test scenario with a higher reward of 9.56

Figure 4.6 Example of the vehicle kinematic model trajectory (blue points) given the road topology (yellow points)

4.3.2 RL Agent Training

In the RIGAA approach, another important step is training a reinforcement learning agent to generate test scenarios. We represent test scenarios as a sequence of discrete scenario elements E. The main goal of the RL agent is to learn to generate sequences of the scenario elements E, such that valid and challenging test scenarios are produced. In the following paragraphs we describe in more detail how we represent the test scenario generation problem as a Markov decision process (MDP), defined by the 5-tuple $< S, A, R, P, \rho_0 >$, and how the RL agent is trained.

State representation. We represent the state as a 2D integer array, such as described in Table 8.1. Each column corresponds to some discrete part of the test scenario. At initialization, the agent starts from a 2D array, where the first column (first scenario element) contains attribute values randomly sampled from the allowable range of values. The rest of cells are initialized with zero values. In the next steps, the goal of the agent is to learn to select such attribute values A_{1em}, A_{2em}, A_{nem} for the elements E_m that result in producing a challenging test scenario.

For an autonomous robot, the state corresponds to a 2D map with obstacles. The state array size is 40 x 3 elements, where 3 corresponds to the number of attributes describing each scenario element (wall type, wall position, wall size). For an autonomous vehicle LKAS system, the state corresponds to a 2D array defining the road topology. The size of the state array size is 30 x 3. Here 30 corresponds to maximum number of scenario elements i.e. road segments. From our experience, the generated test scenarios rarely exceed 30 road segments, that is why we fixed this number to 30. If the scenario contains less, than 30 elements, the rest is padded with zeros. The second dimension corresponds to the number of attributes, which is three (road segment type, road segment length and turning angle).

Action space. At each time step i, the agent chooses an action that defines the attribute values of a scenario element E_i. The action is multidimensional, with the number of dimensions equal to the number of attributes. For both test generation agents, the action space has three dimensions. For the autonomous robot, the first dimension corresponds to the type of obstacle, the second to the obstacle's position, and the third to the obstacle's size. The resulting action space was of size $2x7x37$. For the autonomous vehicle, the first dimension corresponds to the type of road segment, the second to the road segment's length, and the third to the turning angle of the road. The resulting action space was of size $3x25x35$.

Reward function. During training, the RL agent receives a reward $r_t \in \mathbb{R}$ every time it executes an action a_t. We define the following reward function:

$$r_i = \begin{cases} R_s + R_1 + R_2 + R_3 & \text{if test scenario is valid,} \\ R_4 & \text{if test scenario is invalid,} \end{cases} \tag{4.3}$$

where R_s represents the surrogate reward of the agent, R_1 denotes the reward improvement score, R_2 provides a bonus reward for discovering a good test scenario and R_3 provides an additional reward for exploring the search space. R_4 represents a negative reward given for producing an invalid test scenario. The improvement score R_1 is a reward given for an action, that leads to a better test scenario, compared to the previous action. We calculate it as $R_1 = (R_{s_t}$ - $R_{s_t-1})$, where R_{s_t} is R_s at the current step and R_{s_t-1} is the R_s reward in the previous step. Such reward encourages the RL agent to seek actions that result in improved test scenarios. The bonus reward R_2 is an additional reward given to the agent if the surrogate reward R_s it received exceeds a certain positive threshold th. If the threshold value is not reached, we set it to zero. In our experiments, without such a reward, the agent took more time to discover more challenging scenarios, that have high R_s. Reward R_3 for exploring the search space is given when a previously unused scenario element is used e.g., a new road segment configuration, different from the ones used before. This promotes diversity in the created scenarios. We give a negative reward R_4 when the scenario does not satisfy the problem constraints.

For an autonomous robot, the surrogate reward R_0 is defined in Eq. (4.1) as the length of the path it needs to travel to the goal. Intuitively, the longer the path it takes, the more challenging the scenario and the higher risk of the robot failing its mission. We fix the threshold th for obtaining the bonus reward at 110 i.e. the agent should achieve the reward of 110 to get an additional reward R_2. We fix R_2 value equal to R_s. We also give a reward of $R_3 = 10$ if the action taken specifies an obstacle position or size, not used before. We choose the R_4 value to be -100.

For an autonomous vehicle LKAS system, the surrogate reward R_s is defined in Eq. (4.2) and represents the vehicle deviation from the road lane center. The bonus reward of $R_2 = R_s$ is given if the surrogate reward R_s exceeds the value of 5. Additionally, a reward for exploration of $R_3 = 1$ is provided if the action taken specifies a segment length or turning angle that has not been used before. The reward for invalid test scenario R_4 is fixed at -50.

Transition probability function. The transition function P determines the probability of transitioning to a new state given the current state and action. In our case, this function is deterministic. Every time the action taken defines specific attribute values of the next scenario element.

Initial state distribution. At initialization, the agent starts from a state i.e. 2D array, where the first column (first scenario element) contains randomly sampled attribute values.

Episode termination. We terminate the episode when the agent provides values for all the scenario elements or after it produces an invalid scenario. For an autonomous robot, we had 40 scenario elements. One of them was initialized at the start, so the agent had to complete 39 steps more to set all the scenario elements and complete the episode. The episode could also be terminated if the agent produced a map not satisfying the constraints. For an autonomous vehicle LKAS system, we set the maximum number of scenario elements to be 30. The episode was terminated either after 30 steps or when the agent produced an invalid road topology.

Reinforcement learning algorithm. For training, we use the state-of-the-art Proximal Policy Optimization (PPO) algorithm [169], implemented in the Stable Baselines framework [170]. PPO is a popular algorithm in the RL community as it requires less hyperparameter fine-tuning, shows good training stability as well as increased sample efficiency [171]. Years following its creation, PPO remains a state-of-the art DRL algorithm [172]. It was included and benchmarked in CleanRL library [173], outperforming other popular RL algorithms, such as DDPG, DQN, TD3 on several environments. Moreover, PPO supports a discrete action space, which is important in our implementation. PPO is an actor-critic algorithm that consists of two key components: the actor and the critic neural network. The actor network is responsible for selecting actions based on the current state, while the critic network evaluates the chosen actions and provides feedback on their quality. This feedback is then used to guide the actor in adjusting its decision-making process [174]. In our study, we use the default multilayer perceptron (MLP) architecture for both the actor and critic, which consists of two layers of 64 neurons. In this case, an observation (which is also an input to the actor network) is a one-dimensional vector that corresponds to the flattened state array. We also experimented with using a convolutional neural network (CNN) for the agent actor-critic architecture, where the observations were images. However, we found that using a multi-layer perceptron (MLP) based architecture resulted in better and more diverse results.

We used the default hyperparameters suggested by the framework, as they showed good stability of training. These hyperparameters include the learning rate of 0.0003, 2048 steps and batch size of 64. One hyperparameter we found was important to fine-tune is the entropy coefficient, which plays an important role in balancing the exploration and exploitation trade-off during training. By default, it is set to 0, resulting in minimal exploration of the action space. In our experiments, with the entropy coefficient set to 0, the agent tended to exploit a particular sequence of actions and the generated scenarios were of low diversity. However,

if this value was set too high, the training became unstable, with occasional failures of the agent to find a good policy and outperform the random generator. Empirically, we fixed the entropy coefficient value of 0.005 for both maze and road topology generation agents, which allowed producing diverse, but at the same time challenging test scenarios.

4.3.3 Genetic Algorithm Implementation

A key component of RIGAA is the use of evolutionary search, specifically the NSGA-II genetic algorithm. We selected the NSGA-II evolutionary algorithm, as one of the commonly used algorithms for search-based software engineering tasks [175]. In this subsection, we describe the individual representation, objective functions, crossover, and mutation operators used in our implementation of NSGA-II.

Individual representation. In our representation of the individuals, or chromosomes, we use a two-dimensional array as shown in Table 8.1. Each chromosome corresponds to an array with columns representing the scenario elements and rows containing attributes describing them. For autonomous robots, the individual size is fixed at 40 x 3. For autonomous vehicles, the individual size is flexible and is of $Nx3$, subject to the constraint that the resulting test scenario must fit within the bounds of the map (200 m x 200 m).

Fitness function. Our framework utilizes a multi-objective search algorithm, NSGA-II [90], to optimize the test suite for both scenario fault revealing power and diversity. The first objective function, F_1, measures the extent to which the system falsifies its requirements when executing a given scenario. We can define the requirements for the system in terms of Metric Temporal Logic (MTL), which is common in CPS testing literature [69]. MTL is a formalism that enables system engineers to express complex design requirements in a formal logic [176]. Then we are interested in measuring the degree to which this requirement can be satisfied. To this end, in CPS testing, a robustness metric is used, which aims to provide a numerical evaluation of the degree of the requirement satisfaction [78]. We evaluate the objective F_1 according to the following equation:

$$F_1 = \rho_{\text{safe}}(\mathcal{G}_{[0,T]}(\phi)), \tag{4.4}$$

where $\mathcal{G}_{[0,T)}$ denotes that globally i.e. at each time interval $t \in [0, T]$, the condition ϕ within the parentheses, which must always hold true. The function ρ_{safe} is a robustness function that returns a value representing the degree to which the MTL property is satisfied. If the property is not satisfied, it returns a negative real value corresponding to the degree to which the property is violated. If the property is satisfied, it returns a positive value. The ϕ function

is problem-specific and can be tailored to the specific system being tested. For example, in the case of an autonomous robot, we define the following function ϕ:

$$\phi = \neg\pi_{coll} \wedge \neg\pi_{fall}, \tag{4.5}$$

where π_{coll} is an event of colliding with an object and π_{fall} is an event of falling down. The ϕ function states that the robot should not collide with objects and should not fall. Then the robustness function ρ_{safe} evaluates the inverse of the proportion of waypoints the robot completed to all of the waypoints that define the path to the target location. It returns the negative value corresponding to the inverse of the proportion of completed waypoints. The less waypoints were completed, the lower will be the value of ρ_{safe} (value will be more negative). Intuitively, the fewer waypoints are completed by the robot, the less successful it is in its mission of reaching the target. For instance, an F_1 fitness value of 5 indicates that the robot has completed only $1/5 = 0.2$ or 20% of all the waypoints in the target path. The goal of our search algorithm is to find a test scenario minimizing the number of waypoints completed. We average the F_1 function values over 3 executions of the agent in the same scenario, due to the stochasticity of the agent's behavior.

For an autonomous vehicle LKAS system, ϕ is defined as follows:

$$\phi = |dist(t)| \leq \delta, \tag{4.6}$$

where $dist$ is a Euclidean distance from the nearest road center point and δ is the maximum allowable deviation from the road center. The ϕ function states that the vehicle should stay within certain road lane boundaries. The robustness function ρ_{safe} returns the maximum percentage of the vehicle area, that went out of the road lane taken with a negative sign. Therefore, for an autonomous vehicle, the F_1 is evaluated as the maximum percentage of vehicle area that went out of bounds over the scenario execution.

To promote diversity in the test suite, we also evaluate a second objective function, F_2, which measures the novelty of an individual scenario. It is defined as the average Jaccard distance of the scenario to N individuals with the highest value of the first fitness function. In our experiments we used the value of $N = 5$. Jaccard distance is a well known metric for evaluating the dissimilarity between two sets [177]. The value of F_2 objective ranges from 0 to 1. This function is inspired by novelty search algorithms [178], where the solution is compared to the archive of the best solutions. Below we provide more details about calculating the Jaccard distance and the F_2 objective function.

The Jaccard distance, or diversity D, between two test cases is calculated as follows:

$$D = 1 - \frac{|TS_1 \cap TS_2|}{|TS_1 \cup TS_2|} \tag{4.7}$$

Here $|TS_1 \cap TS_2|$ represents the number of scenario elements that are shared between two test scenarios, TS_1 and TS_2 i.e. belong to the intersection set between two test scenarios. Scenario elements are added to the intersection set if their attributes are sufficiently similar, as defined by thresholds th_0, th_1, th_n for attributes A_0, A_1, A_n, respectively. The thresholds should be selected based on the domain knowledge of the problem. Two scenario elements are considered similar if all of their corresponding attributes are similar to one another. We calculate the number of scenario elements in the union of two sets, TS_1 and TS_2, using the Eq. (4.8):

$$|TS_1 \cup TS_2| = |TS_1| + |TS_2| - |TS_1 \cap TS_2| \tag{4.8}$$

where $|TS_1|$ is the number of elements in TS_1, $|TS_2|$ is the number of elements in TS_2, and $|TS_1 \cap TS_2|$ is the number of scenario elements that are common to both TS_1 and TS_2. Finally, to evaluate the F_2 objective of the individual, as mentioned above, we calculate the average diversity between the individual and N=5 individuals with the highest F_1 fitness function in the population, as shown in Eq.(4.9):

$$F_2 = \frac{1}{N} \sum_{i=1}^{N} D(tc_0, tc_i), \tag{4.9}$$

where tc_0 represents the test scenario, for which we evaluate the fitness, and tc_i represents one of the N best scenarios in terms of the F_1 function discovered by the algorithm. We set the N=5, corresponding to the 5 best solutions found in terms of F_1 objective in a given generation of the algorithm. In our implementation, we aim to minimize F_1 and maximize F_2. The Pymoo framework [179] that we use for genetic algorithm implementation, only supports function minimization. As a result, we minimize the negative value F_2 instead.

Crossover operator. We are using a one point crossover operator, which is one of the commonly used operators for variable-length solution representation. An example of applying this operator is shown in Figure 4.7. This operator creates two new test cases by exchanging scenario elements between two existing test scenarios. The crossover operator is applied with probability $CROS$ and parents are simply copied with probability $1 - CROS$.

Mutation operators. We define two mutation operators: exchange M_e and change of vari-

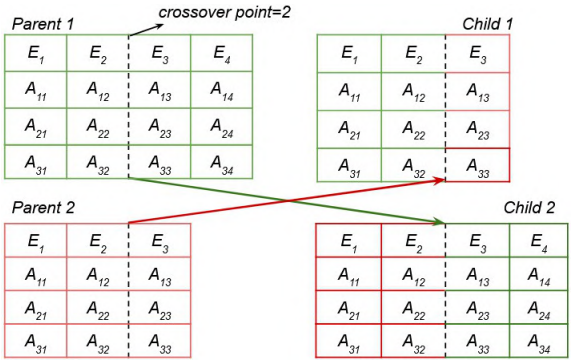


Figure 4.7 An example of the crossover operator functioning, crossover point is selected equal to 2

able operator M_c. For exchange operator, the attributes of the randomly selected scenario elements E_m are exchanged the positions. In change of variable operator, scenario element E_m in a chromosome is randomly selected. Then the value of one of its attributes A_{nem} is changed according to its type and maximum as well as minimum value. Examples of applying these operators are shown in Figure 4.8 and Figure 4.9 respectively. After the crossover, the individuals undergo mutation with probability MUT. If the individual is selected for mutation, one of the mutation operators, M_e or M_c is applied with equal probability. In M_e two genes of the chromosome (scenario elements) are changed, while in M_c only one gene, corresponding to the mutation rate of $\frac{2}{m}$ and $\frac{1}{m}$ respectively. As the probability of applying the operators is the same, the average mutation rate would be $(\frac{1}{m} + \frac{2}{m}) \cdot \frac{1}{2} = 1.5\frac{1}{m}$. To make this value closer to the recommended range of mutation rate [40] of $\frac{1}{N_{pop}} to \frac{1}{m}$ (N_{pop} is the population size), we fix the probability for individuals to undergo a mutation at MUT= 0.4 [157], resulting in an average mutation rate of $1.5 \cdot 0.4 \cdot \frac{1}{m} = 0.6 \cdot \frac{1}{m}$.

Duplicate removal. At each iteration, we remove the individuals which have small diversity D (see Eq. 4.7) between them. We fix the D threshold to be 0.2. If the diversity between the two individuals is less than 0.2, one of them is removed.

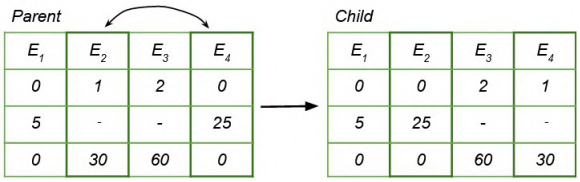


Figure 4.8 An example of the exchange mutation operator, where segments 2 and 4 are exchanged

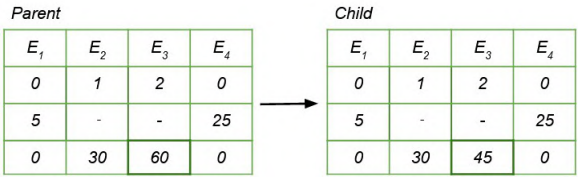


Figure 4.9 An example of the change of variable mutation operator, where the value of the third attribute of the third segment is changed

4.3.4 RIGAA Algorithm Implementation

The main steps of RIGAA are outlined in Algorithm 1, and the value ranges of the main hyperparameters are shown in Table 4.5. When running RIGAA in simulation, we recommend using smaller population sizes on the order of 40-50 individuals. When surrogate function is used for evaluation, bigger population size can be used i.e. 100-200 individuals. In our experiments, we used the crossover rate of 0.9 and mutation probability of 0.4. N_{TC} hyperparameter corresponds to the number of the best solutions to be selected from the final generation, and it cannot be bigger, than population size N_{pop}.

Table 4.5 RIGAA algrithm hyperparameters

Parameter	Description	Values
N_{pop}	Population size	40 - 200
$CROS$	Crossover rate	0.5 - 0.9
MUT	Mutation probability	0.4 - 0.6
N_{TC}	Test suite size	20 - 100
ρ	Percentage of RL agent generated individuals	0.2 - 0.4

As every evolutionary algorithm, RIGAA starts by creating an initial population of the size N_{pop}. Differently from existing initial population sampling techniques, in RIGAA some percentage ρ of individuals is generated by a pre-trained RL agent and the other by a random generator. After initial population generation, RIGAA is following the steps of the NSGA-II algorithm with objectives that we formulated in the sub-section 4.3.3. At each iteration, tournament selection is used to choose the individuals for crossover and mutation, which are then performed on individuals with probabilities $CROS$ and MUT respectively. The search is continued until a termination criterion is met. Finally, N_{TC} best individuals are selected from the final population to be added to the test suite. In our case, we used the number of evaluations or the running time as the termination criteria.

In the following section, we assess the performance of RIGAA by comparing its performance in terms of number of failures and their diversity (sparseness) with other algorithms. Specif-

ically, we compare it with random search, our previous implementation of NSGA-II-based search as well as the RL generation without evolution. We also compare RIGAA to other available baselines and evaluate different configurations of RIGAA.

Algorithm 1 RIGAA: Reinforcement learning Informed initialization Genetic Algorithm for Autonomous system testing

Set hyperparameters N_{pop}, $CROS$, MUT, N_{TC}, ε,
Load a pre-trained RL model
Initialize population P:
while $N \neq N_{pop}$ **do**
    Generate an individual with the pre-trained RL model with probability ε
    Generate an individual randomly with probability $1 - \varepsilon$
    $N = N + 1$
end while
Run the genetic algorithm:
Evaluate objectives F_1, F_2 over population P
while not (T) **do** # T - a termination criterion
    Select individuals for mating from $P \rightarrow P'$ based on tournament selection
    Apply crossover with probability of $CROS$ $P' \rightarrow P''$
    Apply mutation with probability MUT $P'' \rightarrow P'''$
    Evaluate fitness objectives of the offsprings P'''
    Construct new population$(P \cup P''') \rightarrow P^*$ from parents and offsprings
    Select the individuals based on crowding distance from P^* of the size of N_{pop}, $P^* \rightarrow P$
end while
Add N_{TC} best individuals from the final generation to the test suite.

4.4 Evaluation

In this section, we formulate and answer the research questions related to the performance of RIGAA. Our primary objective is to assess the extent to which the RL-based initialization enhances the performance of a multi-objective evolutionary algorithm (MOEA) in comparison to the random initialization. Additionally, we aim to explore which RIGAA configuration yields the optimal performance.

For comparing the algorithms, we assessed the following metrics: the average number of revealed failures F_{av} and their sparseness S_{av} [81], which corresponds to the diversity of the failures. We calculated the sparseness as the average distance between all the pairs of the failed test scenarios according to the following equation:

$$S_{av} = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} Dist(tc_i, tc_j), \tag{4.10}$$

where $\frac{N(N-1)}{2}$ is the total number of pairs of the failed test cases, N is the number of revealed failures, $Dist$ is a distance metric evaluated between two test scenarios tc_i and tc_j. For the autonomous robot case study $Dist$ was evaluated as the Jaccard distance according to Eq. (4.7). For the autonomous vehicle, we calculated $Dist$ as the weighted Levenshtein distance [180] between road segments near the location where the failure happened (car went out of the bounds) as suggested by Riccio and Tonella [81]. This metric was also used at the SBST 2021 tool competition [160].

For the autonomous robot case study, we executed each test scenario three times due to the stochastic nature of the agent behavior. We considered a test as failed, if in all the three runs the agent completed less than 50 % of the expected waypoints on the path to the target. For the autonomous vehicle, a test case is considered failed if more than 0.85 % of the vehicle goes out of the bounds. We used the 0.85% threshold as it was adopted in the related testing tool competitions [160], [155], [181].

For all research questions, where RIGAA is evaluated, we used the RIGAA configuration with a population size N_{pop} of 40, crossover rate $CROS$ 0f 0.9 and mutation probability of MUT of 0.4. We repeated all evaluations a statistically significant number of times, i.e. at least 10 when using the simulator. For all results, we report the non-parametric Mann-Whitney U test with a significance level of $\alpha = 0.05$, as well as the effect size measure in terms of Cliff's delta [182, 183]. We run our experiments on a PC with a 16-core AMD Ryzen 7 4800HS CPU @ 2.90 GHz, 16 GB of Memory, and an NVidia GeForce GTX 1660 GPU @ 6GB.

4.4.1 Research Questions

In the first part of our experiments, we evaluate the performance of RIGAA in terms of its ability to falsify simulator-based autonomous robotic systems. We evaluate the performance of RIGAA by measuring the average number of failures (F_{av}) and their sparseness (S_{av}) revealed by the generated test scenarios for an autonomous robot and an autonomous vehicle. We compare RIGAA to the available baselines. In the second part, we aim to identify the optimal configuration of RIGAA and compare it with the baseline MOEA (MOEA with random initialization). We test RIGAA with different values of the initial percentage of RL-produced solutions, as well as with two different MOEAs. Finally, we evaluate the trained RL agent performance and cost, comparing it to the random generator. Our expectation is that the RL agent will produce scenarios of higher fitness than a random generator while maintaining a similar level of diversity in the produced scenarios.

We formulate the following research questions:

1. **RQ1** (Usefulness of RIGAA for simulator-based testing of autonomous robotic systems.). *To what extent can RIGAA improve the identification of diverse failures of autonomous robotic systems compared to baseline approaches?* In *RQ*1 we compare the RIGAA performance in terms of number of failures found and their diversity to baseline approaches such as Random generation, NSGA-II algorithm without RL-based initialization and RL agent without evolution. Additionally, for the autonomous vehicle case study, we compare RIGAA with state-of-the-art tools for vehicle LKAS system testing as Frenetic [82], WOGAN [184], AmbieGen [185] and CRAG [186].

2. **RQ2** (Selecting the ρ hyperparameter of the RIGAA algorithm). *What is the effect of different values of ρ on the number of revealed failures and their sparseness?*

   The aim of RQ2 is to investigate the impact of the percentage ρ of RL-produced solutions in the initial population on the number and sparseness of the revealed failures. We evaluate RIGAA using six different values of the ρ hyperparameter, namely $\rho \in [0, 0.2, 0.4, 0.6, 0.8, 1]$, and recommend the optimal value based on the resulting failure quantity and sparseness.

3. **RQ3** (Comparing RIGAA and randomly initialized MOEA). *How do different configurations of RIGAA compare to each other and baseline MOEA?*

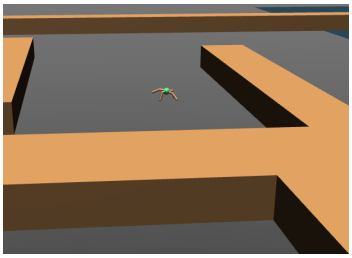
   In *RQ*2, our objective is to evaluate the effectiveness of the RL-based initialization in guiding an evolutionary algorithm and improving its performance in terms of the number of the failures revealed and their sparseness. For this purpose, we compare different RIGAA configurations based on two MOEAs, namely NSGA-II [47] and another multiobjective optimization algorithm SMS-EMOA [187], to the corresponding MOEA configurations with the random initialization. We selected NSGA-II as it is one of the most popular MOEA to be used for search-based software engineering problems [175]. We select SMS-EMOA as another type of MOEA, having a different selection algorithm. It differs from NSGA-II in promoting the dominated hypervolume rather than the crowding distance of solutions. SMS-EMOA allows finding solutions with a good compromise of the search objectives and a well-distributed Pareto front, which is promising for test scenario generation as we aim to maximize the diversity of solutions i.e. the test scenarios.

4. **RQ4** (Comparing the performance of the RL-based test generator and random test generator). *How do the performance metrics of the RL agent and random generator compare in terms of the average fitness and diversity of the test scenarios produced, as well as the generation time?*

In $RQ1$ we aim to assess whether the trained RL agent produces scenarios of higher fitness, than the random parameter initialization. We evaluate the fitness of the solutions based on the fitness of the scenarios when applied to the simulator-based models of the systems under test. Moreover, we aim to evaluate if the RL agent can generate solutions with high diversity. If it is the case, we can argue that the RL agent can indeed provide a better initialization for the evolutionary algorithm, than a random generator.

4.4.2 Subjects of the Study

In our study, we consider two simulator-based autonomous robotic systems: an autonomous ant robot in the Mujoco simulator [188] as well as an autonomous vehicle in BeamNG simulator [148].



(a) Autonomous ant robot in Mujoco

(b) Autonomous vehicle in BeamNG

Figure 4.10 Autonomous ant and autonomous vehicle systems used in our study

The autonomous ant robot, shown in Figure 5.6a is an 8-DoF (degrees-of-freedom) "Ant" quadruped robot designed for research on RL-based agent by Schulman et al. [189]. This robot consists of one torso (free rotational body) with four legs attached to it with each leg having two links. The goal of the RL algorithm is to coordinate the four legs to move in the forward (right) direction by applying torques on the eight hinges connecting the two links of each leg and the torso. We utilize the Ant robot pre-trained with an RL policy from the D4RL benchmark [156] for locomotion. Additionally, we employ a maze environment provided by the authors of D4RL benchmark, which requires the autonomous ant to navigate towards goal locations. In our experiments, we modified the environment to create larger maze maps and evaluate the behavior of the agent.

As the second test subject, we used an autonomous driving agent BeamNG.AI, built-in in the BeamNG.tech [148] simulator, shown in Figure 5.6b. The goal of the driving agent is to navigate a given road, without going out of its bounds. It has knowledge of the geometry

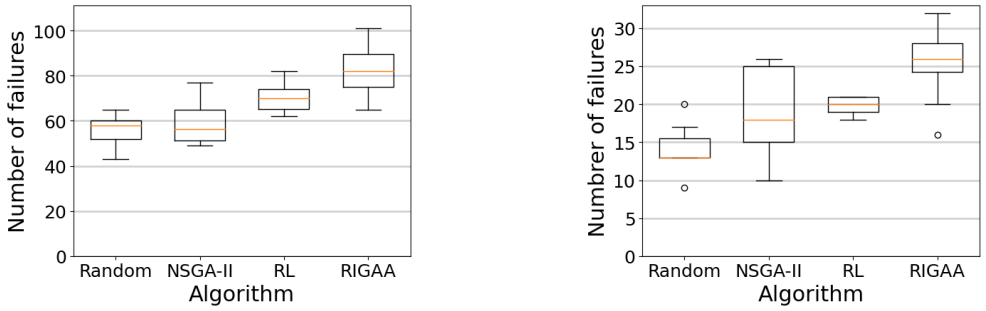
of the entire road and uses a complex optimization process to plan trajectories that keep the ego-car as close as possible to the speed limit while staying inside the lane as much as possible.

4.4.3 Results

In this subsection, we present our experimental results for each RQ formulated above.

RQ1 – Usefulness of RIGAA for simulator-based testing of autonomous robotic systems.

In this research question, our aim is to assess the ability of RIGAA to generate diverse and challenging test scenarios for an autonomous ant robot in the Mujoco simulator and an autonomous lane keeping assist system in the BeamNG simulator. We compare RIGAA to baseline NSGA-II, pre-trained RL agent (RL) without evolution and random search (Random) in terms of the average number of revealed failures F_{av} and average sparseness S_{av} of the failures. For the autonomous vehicle we also compare RIGAA with the available open-source tools for LKAS testing. We allocate a two-hour time budget for each run of the algorithms, and we run each algorithm at least 10 times. We used the RIGAA configuration with a population size of 40 and 40 offspring and $\rho = 0.4$. Some examples of the RIGAA detected autonomous ant robot and LKAS failures are shown in the following embedded links with videos: *link to video 1*, *link to video 2*.

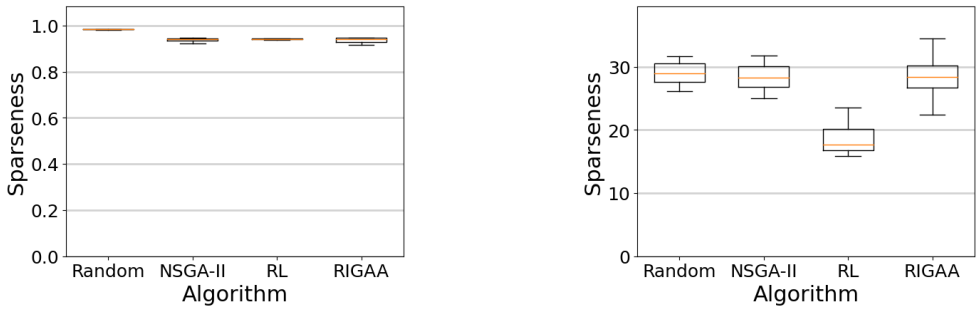


(a) The average number of failures for autonomous ant robot

(b) The average number of failures for autonomous vehicle

Figure 4.11 The average number of failures detected by different algorithms for the simulator-based autonomous robotic systems

The box plots for the average number of failures revealed and their sparseness for the autonomous robot and vehicle are presented in Figure 4.11 and Figure 4.12 respectively. Ta-



(a) The average sparseness of failures of the autonomous ant robot

(b) The average sparseness of failures of the LKAS

Figure 4.12 The average sparseness of failures detected by different algorithms for the simulator-based autonomous robotic systems

Table 4.6 The average number of failures and their sparseness revealed by different algorithms

Problem	Metric	Random	NSGA-II	RL	RIGAA
robot	Failures, F_{av}	56	59.75	70.25	**82.1**
robot	Sparseness, S_{av}	**0.983**	0.937	0.941	0.936
vehicle	Failures, F_{av}	14.125	19	19.833	**25.5**
vehicle	Sparseness, S_{av}	**29.156**	28.813	18.663	28.472

bles 4.6 - 4.8 report he obtained mean values and statistical tests results. We can observe that RIGAA is able to find more failures for every system under test compared to random search, NSGA-II and RL generator. For the autonomous robot, RIGAA finds 16.8 % more failures, than the RL generator, 37.4% more than NSGA-II and 46.6% more than random search. The sparseness of failures detected by the random search is 4.6 % higher than that of the other algorithms. At the same time, there is no significant difference in sparseness between NSGA-II, RL agent and RIGAA revealed failures. From Figure 4.13a we can see that RIGAA converges faster to a bigger number of failures, than the other algorithms.

For the autonomous vehicle, RIGAA finds 28.5% more failures than the RL agent, 34.2 % more than NSGA-II and 80 % more, than random search. The failures revealed by the RL agent without evolution have the lowest sparseness. The difference in sparseness for the other algorithms is insignificant and the value of sparseness is 56.2 % higher, than that of the RL agent. From Figure 4.13, we can observe that RIGAA converges faster to test scenarios that cause the vehicle to go out of the road bounds.

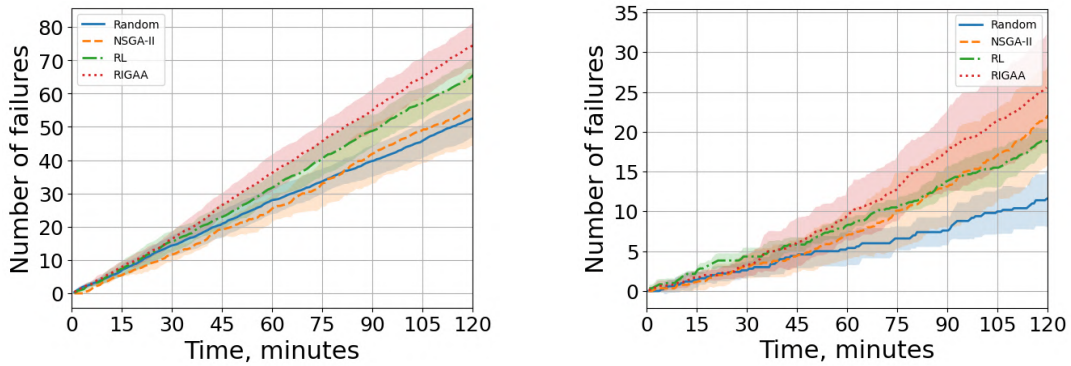
We also compare RIGAA with other state-of-the-art tools for testing autonomous vehicle LKAS system. Unfortunately, to the best of our effort, we have not found reproducible tools for autonomous robot testing with customizable simulation environments. We compare

Table 4.7 Non-parametric test results and effect sizes of the failures found and their sparseness for the autonomous robot case study

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	Random	NSGA-II	0.685	-0.147, S	Sparseness,	Random	NSGA-II	<0.001	1.0, L
F_{av}	Random	RL	<0.001	-0.949, L	S_{av}	Random	RL	<0.001	1.0, L
	Random	RIGAA	<0.001	-0.994, L		Random	RIGAA	<0.001	1.0, L
	NSGA-II	RL	0.154	-0.562, L		NSGA-II	RL	0.683	-0.188, S
	NSGA-II	RIGAA	0.028	-0.8, L		NSGA-II	RIGAA	0.945	-0.05, N
	RL	RIGAA	0.023	-0.65, L		RL	RIGAA	0.897	0.05, N

Table 4.8 Non-parametric test results and effect sizes of the failures found and their sparseness for the LKAS case study

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	Random	NSGA-II	0.041	-0.569, L	Sparseness,	Random	NSGA-II	0.953	0.04, N
F_{av}	Random	RL	0.033	-0.8, L	S_{av}	Random	RL	0.004	1.0, L
	Random	RIGAA	0.002	-0.957, L		Random	RIGAA	0.622	0.171, S
	NSGA-II	RL	0.585	-0.183, S		NSGA-II	RL	<0.001	1.0, L
	NSGA-II	RIGAA	0.004	-0.7, L		NSGA-II	RIGAA	0.815	0.064, N
	RL	RIGAA	0.01	-0.75, L		RL	RIGAA	<0.001	-0.976, L



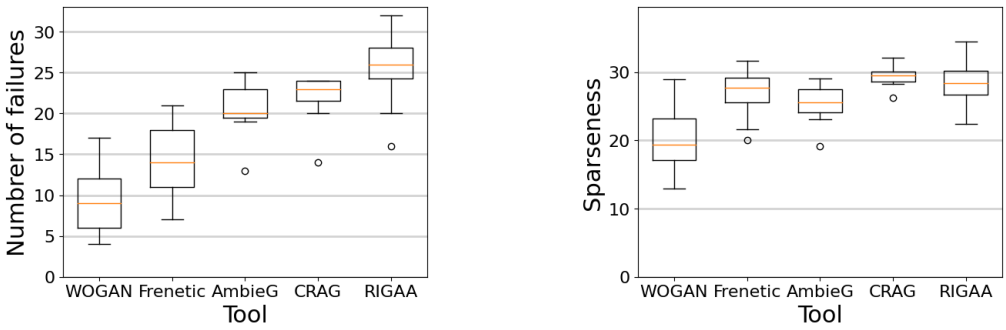
(a) Number of revealed failures over time for autonomous robot testing

(b) Number of revealed failures over time for LKAS testing

Figure 4.13 Number of revealed failures over time for autonomous robot and autonomous vehicle simulator based models

RIGAA with Frenetic tool [82], the winner of the SBST 2021 competition [160], WOGAN tool [184], AmbieGen (AmbieG) [185] (the tool we previously proposed) and CRAG [186] (winner of the SBFT 2023 tool competition [181]). We use the configuration of AmbieGen, where the simulator-based model is used to guide the search, rather, than the simplified objective function.

For the tool comparison, we allocated a fixed time budget of 2 hours to all the tools. We



(a) Number of failures detected by the tools (b) The sparseness of the failures detected

Figure 4.14 Comparing state-of-the-art tools for autonomous vehicle lane keeping assist system testing

Table 4.9 Average number of failures detected by the tools and their sparseness

Metric	WOGAN	Frenetic	AmbieG	CRAG	RIGAA
Failures	9.308	14	20.636	21.714	**25.5**
Sparseness	20.262	27.085	25.57	**29.345**	28.472

then calculated the total number of failures detected, as well as the sparseness of the detected failures. A failure was considered to have occurred when the vehicle went out of the lane bounds by more than 85% of its area. We ran each tool 10 times.

The box plots showing the number of revealed failures, and their sparseness, are presented in Figure 4.14 with corresponding mean values in Table 4.9. Statistical tests are summarized in Table 4.10. We can see that RIGAA outperforms the other tools with a large effect size in terms of the number of revealed failures. Within a two-hour execution budget, RIGAA discovers 15.5% more failures than CRAG, 23.5% more failures than AmbieGen, 82.1 % more, than Frenetic and 174 % more, than WOGAN. CRAG finds the failures with the highest average sparseness of 29.34, but there is no significant difference with the sparseness of the failures revealed by RIGAA.

Table 4.10 Statistical testing results and effect size for LKAS testing tools

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures	WOGAN	Frenetic	0.002	-0.613, L	Sparseness	WOGAN	Frenetic	<0.001	-0.772, L
	WOGAN	AmbieG	<0.001	-0.979, L		WOGAN	AmbieG	0.008	-0.65, L
	WOGAN	CRAG	<0.001	-0.978, L		WOGAN	CRAG	<0.001	-0.934, L
	WOGAN	RIGAA	<0.001	-0.989, L		WOGAN	RIGAA	<0.001	-0.813, L
	Frenetic	AmbieG	<0.001	-0.784, L		Frenetic	AmbieG	0.115	0.329, S
	Frenetic	CRAG	0.001	-0.847, L		Frenetic	CRAG	0.049	-0.488, L
	Frenetic	RIGAA	<0.001	-0.941, L		Frenetic	RIGAA	0.294	-0.202, S
	AmbieG	CRAG	0.311	-0.299, S		AmbieG	CRAG	0.003	-0.818, L
	AmbieG	RIGAA	0.004	-0.695, L		AmbieG	RIGAA	0.046	-0.481, L
	CRAG	RIGAA	0.022	-0.633, L		CRAG	RIGAA	0.488	0.204, S

Summary of RQ1: RIGAA outperforms the RL agent without evolution by revealing 16.8% more failures for an autonomous ant in the Mujoco simulator, and 28.5 % more failures for an autonomous vehicle in the BeamNG simulator. Moreover, the sparseness of the failures revealed by the RL agent for the LKAS was lower with a large effect size, than that of RIGAA, which confirms the benefits of evolving initial solutions produced by the RL agent. RIGAA also outperforms the baseline NSGA-II algorithm with random initialization with a large effect size for both case studies. The difference in sparseness between RIGAA and NSGA-II produced solutions is negligible. RIGAA outperforms the state-of-the-art LKAS testing tools in terms of the number of revealed failures in 2 hours with a large effect size.

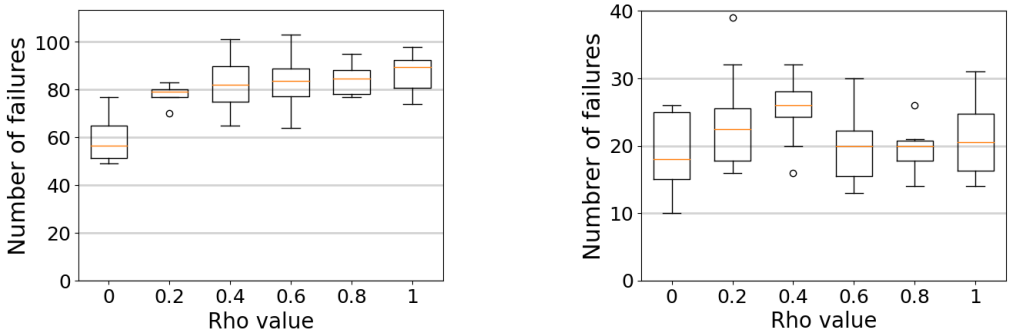
RQ2 - Selecting the ρ hyperparameter of the RIGAA algorithm

In RIGAA the percentage of the initial population generated by the RL agent is defined by the ρ hyperparameter. In this RQ, we investigate the impact of different ρ values on the number and sparseness of the failures revealed.

We inserted solutions produced by the trained RL agent, as a percentage ρ of the population size, where ρ ranges from 0 to 1 in increments of 0.2, i.e. $\rho \in [0, 0.2, 0.4, 0.6, 0.8, 1]$. The rest of the individuals (not RL produced) were generated randomly, by sampling the values uniformly from the allowed ranges. We performed 10 runs of the algorithm in a simulated environment for each configuration with a duration of 2 hours each. For both case studies we used the population and offspring size of 40. The box plots depicting the average number of failures F_{av} and their sparseness S_{av} for each configuration for both the autonomous robot maze and autonomous vehicle road topology generation are presented in Figures 4.15 - 4.16. Tables 4.11- 4.13 show the corresponding mean values and statistical tests results.

Table 4.11 Number of failures and their sparseness for different values of ρ

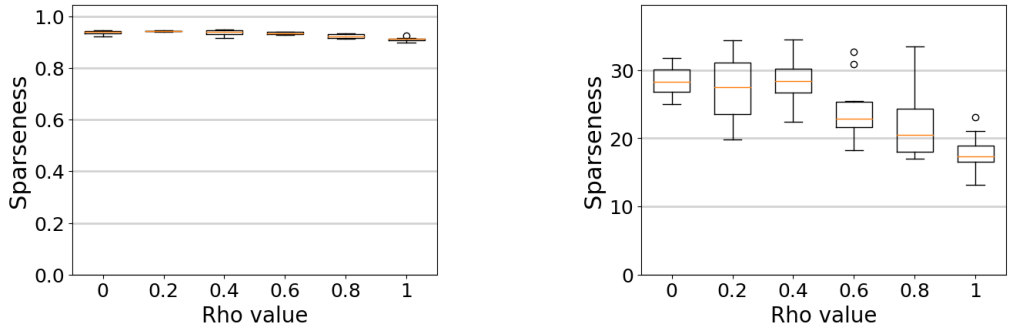
Problem	Metric	0	0.2	0.4	0.6	0.8	1
robot	Failures, F_{av}	59.75	77.75	82.1	82.75	84.5	**87**
robot	Sparseness, S_{av}	0.937	**0.942**	0.936	0.934	0.923	0.91
vehicle	Failures, F_{av}	19	23.125	**25.5**	20.083	19.667	21.3
vehicle	Sparseness, S_{av}	28.49	27.363	**28.472**	24.015	22.372	17.797



(a) Average number of revealed failures for an autonomous robot for different values of ρ

(b) Average number of revealed failures for an autonomous vehicle for different values of ρ

Figure 4.15 Average number of revealed failures for different values of ρ



(a) Average failure sparseness for an autonomous robot

(b) Average failure sparseness for an autonomous vehicle

Figure 4.16 Average failure sparseness for different values of ρ

For the autonomous robot, all the configurations, where some percentage of RL produced individuals was inserted, produce more failures with a large effect size, than when no RL individuals were inserted ($\rho = 0$). There is no statistically significant difference between the number of revealed failures for the configurations with ρ from 0.2 to 1. At the same time,

Table 4.12 Non-parametric test results and effect sizes for the autonomous robot for different values of ρ

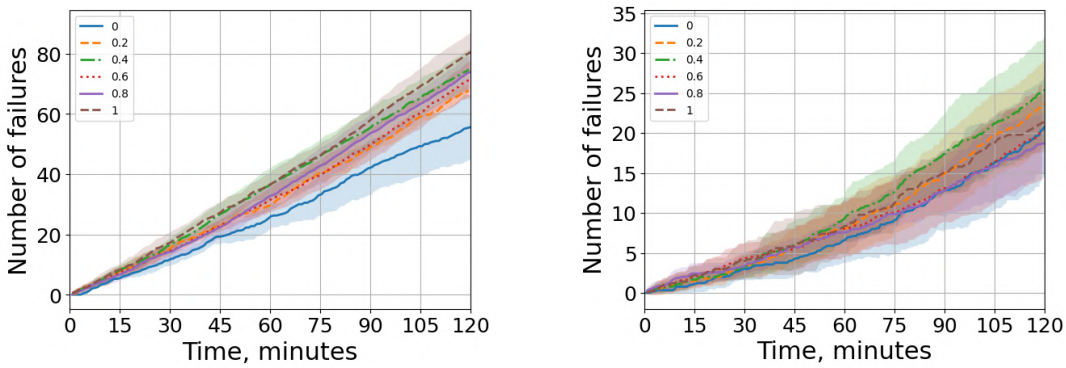
Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	0	0.2	0.059	-0.875, L	Sparseness	0	0.2	0.486	-0.375, M
F_{av}	0	0.4	0.028	-0.8, L	S_{av}	0	0.4	0.945	-0.05, N
	0	0.6	0.016	-0.875, L		0	0.6	0.461	0.312, S
	0	0.8	0.01	-0.969, L		0	0.8	0.048	0.75, L
	0	1	0.017	-0.906, L		0	1	0.008	0.938, L
	0.2	0.4	0.478	-0.275, S		0.2	0.4	0.839	0.1, N
	0.2	0.6	0.349	-0.375, M		0.2	0.6	0.008	0.938, L
	0.2	0.8	0.348	-0.375, M		0.2	0.8	0.004	1.0, L
	0.2	1	0.147	-0.562, L		0.2	1	0.004	1.0, L
	0.4	0.6	1	-0.013, N		0.4	0.6	0.36	0.275, S
	0.4	0.8	0.504	-0.2, S		0.4	0.8	0.027	0.625, L
	0.4	1	0.306	-0.3, S		0.4	1	<0.001	0.925, L
	0.6	0.8	0.916	-0.047, N		0.6	0.8	0.021	0.688, L
	0.6	1	0.4	-0.266, S		0.6	1	<0.001	1.0, L
	0.8	1	0.634	-0.156, S		0.8	1	0.007	0.781, L

Table 4.13 Non-parametric test results and effect sizes for autonomous vehicle for different values of ρ

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	0	0.2	0.068	-0.438, M	Sparseness,	0	0.2	0.414	0.2, S
F_{av}	0	0.4	0.004	-0.7, L	S_{av}	0	0.4	0.815	0.064, N
	0	0.6	0.902	-0.06, N		0	0.6	0.003	0.92, L
	0	0.8	0.663	-0.15, S		0	0.8	0.042	0.633, L
	0	1	0.342	-0.26, S		0	1	<0.001	1.0, L
	0.2	0.4	0.113	-0.344, M		0.2	0.4	0.547	-0.134, N
	0.2	0.6	0.34	0.3, S		0.2	0.6	0.05	0.6, L
	0.2	0.8	0.283	0.312, S		0.2	0.8	0.059	0.542, L
	0.2	1	0.397	0.206, S		0.2	1	<0.001	0.95, L
	0.4	0.6	0.019	0.486, L		0.4	0.6	0.005	0.829, L
	0.4	0.8	0.02	0.679, L		0.4	0.8	0.026	0.643, L
	0.4	1	0.038	0.436, M		0.4	1	<0.001	0.986, L
	0.6	0.8	0.853	0.1, N		0.6	0.8	0.429	0.333, M
	0.6	1	0.58	-0.2, S		0.6	1	0.003	0.92, L
	0.8	1	0.786	-0.1, N		0.8	1	0.093	0.533, L

configurations with $\rho \in [0.8, 1]$ obtained the lowest sparseness value with large effect size. Configuration with $\rho = 0.2$ has the biggest sparseness value with no statically significant difference from $\rho = 0.4$. It's bigger, with large effect size than sparseness for $\rho = 0.6$ configuration.

For autonomous vehicle, the configuration with $\rho = 0.4$ revealed bigger number of failures with large effect size than configurations with $\rho \in [0, 0.6, 0.8, 1]$. It also revealed 10.2 % more failures, than $\rho = 0.2$ configuration, however this difference was not statistically significant.



(a) Number of failures over time for autonomous robot

(b) Number of failures over time for autonomous vehicle

Figure 4.17 Number of failures over time for different values of ρ hyperparameter

Configurations with $\rho \in [0.6, 0.8, 1]$ revealed failures with the lowest sparseness and lower, than configurations $\rho \in [0, 0.2, 04]$ with large effect size. Similarly to the results for the robot case study, we can see that the sparseness value decreases with increasing ρ, for $\rho > 0.4$. At the same time for $\rho \in [0, 0.2, 0.4]$ the sparseness of failures is similar with no statistically significant difference.

The failure convergence plots in Figure 4.17 show that adding RL generated individuals into the initial population allows finding more failures over the given time period. For autonomous robot, all RIGAA configurations show a similar performance in terms of revealed failures in the given time. For the autonomous vehicle, configurations with $\rho \in [0.2, 0.4]$ reveal more failures on average.

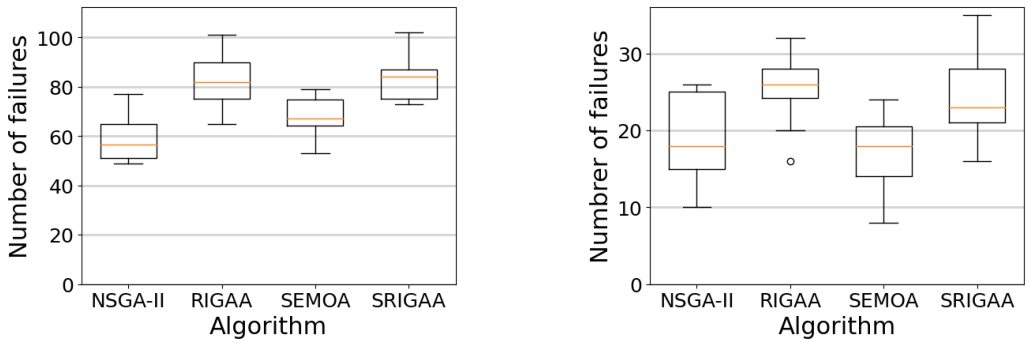
As a result of our analysis, we recommend selecting the ρ hyperparameter from the range of 0.2 to 0.4. A lower percentage of RL produced solutions in the population allows finding more failures, than randomly initialized population for both case studies. At the same time, the sparseness of the failures remains high and with no significant difference from randomly initialized algorithm. Moreover, in evolutionary search literature, it is recommended to have a bigger percentage of randomly produced individuals in the initial population [40].

Summary of RQ2: For autonomous robot problem, for all the tested values of $\rho > 0$, more failures were revealed with large effect size compared to $\rho = 0$ (random initialization). For autonomous vehicle problem, configurations with $\rho \in [0.6, 0.8, 1]$ revealed fewer failures with small effect size, than the random initialization. Configuration with $\rho = 0.4$ revealed the biggest number of failures with big effect size. For both problems, the sparseness of the failures was reducing for the $\rho \in [0.6, 0.8, 1]$. We recommend choosing ρ hyperparameter from the range of 0.2-0.4.

RQ3 – Comparing RIGAA and randomly initialized MOEA

In this research question we compare two configurations of RIGAA, namely RIGAA based on NSGA-II (referred to as "RIGAA") and RIGAA based on SMS-EMOA (referred to as "SRIGAA"), with the corresponding algorithms NSGA-II and SMS-EMOA (referred to as 'SEMOA' in the plots) having a random initialization, and random search. In the baseline MOEA (i.e. NSGA-II and SMS-EMOA), the initial population is generated randomly, whereas in RIGAA and SRIGAA, 40% of the population is produced using a pre-trained RL agent.

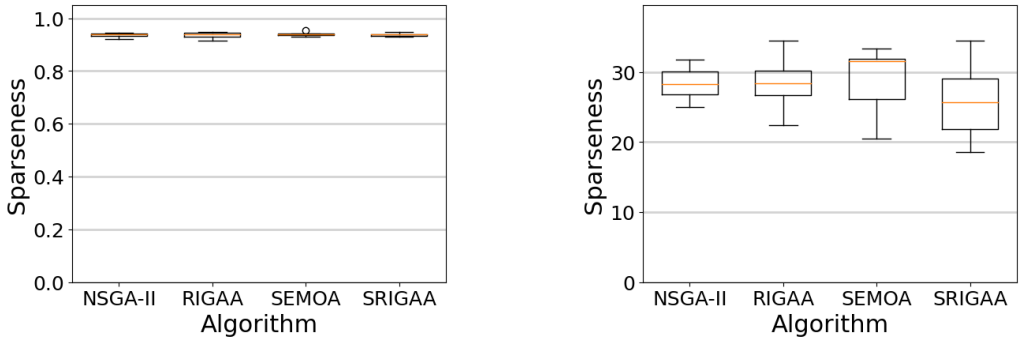
We ran each of the algorithms 10 times in a simulated environment, each run lasted 2 hours. For both case studies we used the algorithms with population and offspring size of 40. The box plots for the average number of failures found F_{av} and their sparseness S_{av} for an autonomous robot and an autonomous vehicle are shown in Figures 4.18- 4.19. Tables 4.14- 4.16 show the corresponding mean values and statistical tests results.



(a) Average number of failures of the autonomous robot

(b) Average number of failures of the autonomous vehicle

Figure 4.18 The average number of revealed failures with different algorithms in 2 hour time budget



(a) Average sparseness of the autonomous robot failures

(b) Average sparseness of the autonomous vehicle failures

Figure 4.19 The average sparseness of the revealed failures with different algorithms

Table 4.14 Average number of failures and their sparseness found by the algorithms

Problem	Metric	NSGA-II	RIGAA	SEMOA	SRIGAA
robot	Failures, F_{av}	59.75	82.1	67.75	**83.889**
robot	Sparseness, S_{av}	0.937	0.936	**0.94**	0.938
vehicle	Failures, F_{av}	18.2	**25.5**	17.091	24.385
vehicle	Sparseness, S_{av}	28.813	28.472	**28.924**	26.06

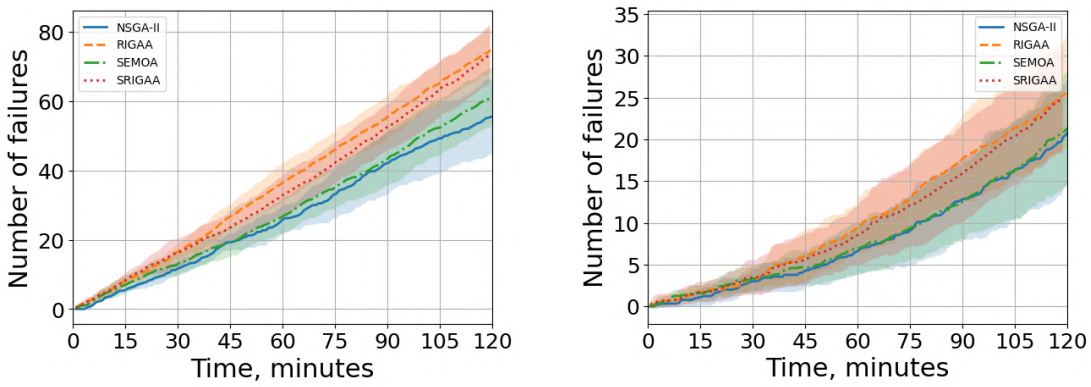
Table 4.15 Non-parametric test results and effect sizes for revealed failures and their sparseness for autonomous robot

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	NSGA-II	RIGAA	0.028	-0.8, L	Sparseness,	NSGA-II	RIGAA	0.945	-0.05, N
F_{av}	NSGA-II	SEMOA	0.233	-0.469, M	S_{av}	NSGA-II	SEMOA	0.933	-0.062, N
	NSGA-II	SRIGAA	0.02	-0.833, L		NSGA-II	SRIGAA	0.94	0.056, N
	RIGAA	SEMOA	0.018	0.675, L		RIGAA	SEMOA	0.897	-0.05, N
	RIGAA	SRIGAA	0.743	-0.1, N		RIGAA	SRIGAA	0.903	0.044, N
	SEMOA	SRIGAA	0.009	-0.764, L		SEMOA	SRIGAA	0.673	0.139, N

Table 4.16 Non-parametric test results and effect sizes for revealed failures and their sparseness for autonomous vehicle

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures,	NSGA-II	RIGAA	0.004	-0.7, L	Sparseness,	NSGA-II	RIGAA	0.815	0.064, N
F_{av}	NSGA-II	SEMOA	0.596	0.145, N	S_{av}	NSGA-II	SEMOA	0.597	-0.145, N
	NSGA-II	SRIGAA	0.045	-0.462, M		NSGA-II	SRIGAA	0.145	0.369, M
	RIGAA	SEMOA	0.001	0.825, L		RIGAA	SEMOA	0.603	-0.13, N
	RIGAA	SRIGAA	0.381	0.203, S		RIGAA	SRIGAA	0.167	0.319, S
	SEMOA	SRIGAA	0.011	-0.622, L		SEMOA	SRIGAA	0.202	0.315, S

Overall, we can see that the algorithms RIGAA and SRIGAA, where some part of the initial



(a) Number of revealed failures over time in autonomous robot problem

(b) Number of revealed failures over time in autonomous vehicle problem

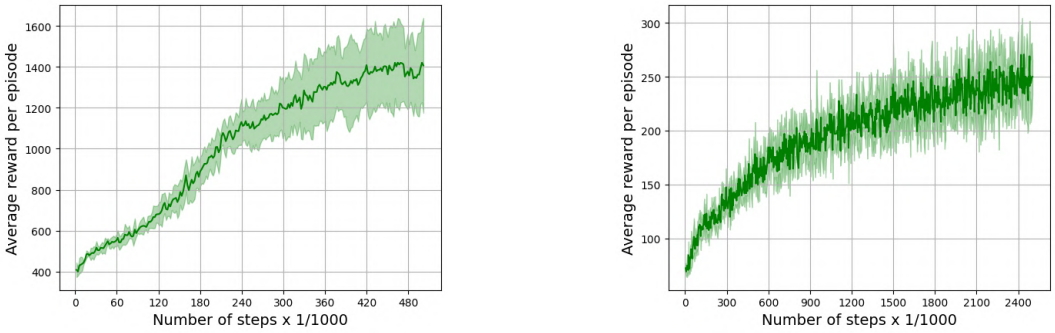
Figure 4.20 Convergence of algorithms for autonomous robot and autonomous vehicle problems

population is generated with an RL agent, are able to find more failures large effect size, than the corresponding algorithms NSGA-II and SMS-EMOA with random initialization. Moreover, the average sparseness of the revealed failures is not statistically different. From Figure 4.20 we can see that RIGAA and SRIGAA converge to bigger number of revealed failures faster, than NSGA-II and SMS-EMOA. We have not observed a statistically significant difference in the performance of RIGAA and SRIGAA.

Summary of RQ3: For both considered multi-objective algorithms, NSGA-II and SMS-EMOA, RL-based initialization allows to reveal more failures with a large effect size, compared to random initialization. No statistically significant difference between RIGAA and SRIGAA was observed. For both problems, the average sparseness of the revealed failures is not statistically different.

RQ4 – Comparing the performance of the RL-based test generator and random test generator

We trained two types of RL agents: one to generate virtual mazes for autonomous robot testing, and the other to generate road topologies for vehicle LKAS system testing. To demonstrate the stability of our training approach, we trained five different RL agents of each type. Figure 4.21 illustrates the mean reward per episode during training, averaged across the five agents. We used a batch size of 64, a learning rate of 0.003 and an entropy coefficient of 0.005 with all other hyperparameters set to their default values as specified in the

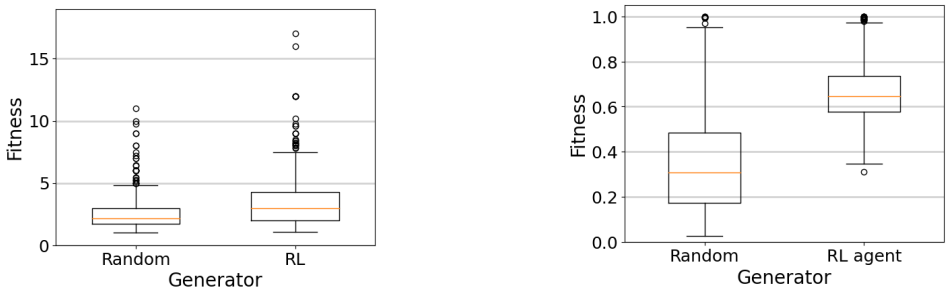


(a) Autonomous robot (b) Autonomous vehicle

Figure 4.21 Average reward per episode during RL agents training

Stable Baselines 3 implementation [163]. We trained the agents for autonomous robot testing for 500,000 steps with the average training time of 13.5 hours and the agents for autonomous vehicle testing for 2,500,000 steps with the average training time of 12 hours. After training the RL agents, we selected one of them to compare its performance to that of a random generator. All the five trained agents achieved similar levels of test scenario F_1 fitness and diversity. We acknowledge that the training time is substantial, however it is only required to be performed once. Moreover, it can be reduced if parallelization techniques, such as running multiple training RL environments at the same time [190], are employed. In addition, it is proportional to the time needed to evaluate the reward. If a fast-to-compute reward function is used, the training time can be drastically reduced. In this training procedure, we did not use any parallelization techniques to speed up the RL agent training, apart from training on a single GPU (NVIDIA GeForce GTX 1660 GPU @ 6GB).

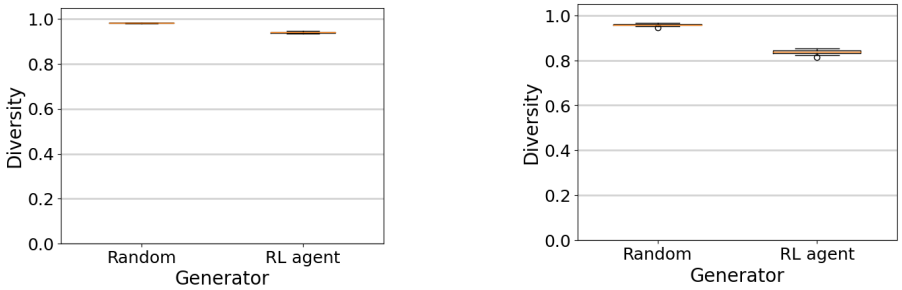
As the next step, we evaluated the ability of the RL agent to produce tests with a higher F_1 fitness in simulator, than the random generator. This requirement is important, as the solutions with higher F_1 fitness are more likely to be selected to be evolved by the search algorithm. We report the absolute value of the F_1 function (defined in Eq.(4.4)) for the better visualization purposes, where a higher value corresponds to a better F_1 fitness. We also evaluate the diversity of the produced solutions as the average diversity metric D (Eq.(4.7)) calculated between all pairs of the generated tests. We generated 10 test suites with 30 test scenarios each. The box plots of the average F_1 fitness and diversity of the test scenarios are shown in Figure 4.22 and Figure 4.23. Table 4.17 summarizes the obtained values for the average test suite F_1 fitness and diversity for the considered problems and generators. We can see that the differences in the F_1 fitness values are statistically significant for both systems under test. The RL agent produces 34.5 % fitter scenarios for an autonomous ant robot and



(a) Average scenario F_1 fitness for an autonomous ant robot in the simulator

(b) Average scenario F_1 fitness for an autonomous vehicle in the simulator

Figure 4.22 Average F_1 fitness of the test scenarios for autonomous robot and vehicle LKAS system in the simulators



(a) Average diversity of the test scenarios for autonomous robot

(b) Average diversity of the test scenarios for the autonomous vehicle

Figure 4.23 Average diversity of the test scenarios produced randomly (left box) and by the RL agent (right box)

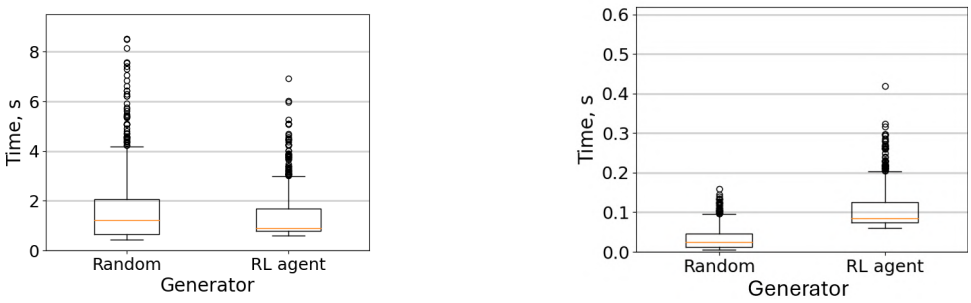
88.5 % fitter scenarios for an autonomous vehicle LKAS system in terms of F_1 objective. The random agent produces 4.46 % more diverse mazes and 14.3 % more diverse road topologies. Although the random generator creates more diverse road topologies, we find the diversity value of 0.839 of the RL generator is sufficiently high, indicating that, on average, each test scenario within a test suite differs by approximately 83.9% from one another.

To evaluate the generators in terms of generation efficiency, we measure the average time taken to produce one solution while creating the test suites that we previously evaluated. The box plots with the generation time of the two RL agents and random generator are shown in Figure 4.24. The corresponding results are reported in Table 4.18.

As we can see, there is a negligible difference between the random and RL-based test genera-

Table 4.17 Results for comparing the test generators in terms of diversity and F_1 fitness function evaluated in simulation

Problem	Metric	Random	RL agent	p-value	Effect size
robot	F_1	2.729	3.671	<0.001	0.243 S
robot	D	0.982	0.94	<0.001	1.0 L
vehicle	F_1	0.35	0.66	<0.001	0.747 L
vehicle	D	0.959	0.839	<0.001	1.0 L



(a) Robot test scenario generation (b) Vehicle test scenario generation

Figure 4.24 Average time to generate one test scenario by different generators

Table 4.18 Results for comparing test case generation time

Problem	Metric	Random	RL agent	p-value	Effect size
robot	mean generation time, s	1.627	1.421	0.24	0.032, N
vehicle	mean generation time, s	0.034	0.106	<0.001	0.886, L

tors for the autonomous robot. The RL-based test generator for the autonomous vehicle takes on average 72 ms more to produce a test scenario, which is also an insignificant overhead, given that the initial population size would rarely exceed 150 individuals.

Summary of RQ4: The RL agents for both the autonomous ant robot and autonomous vehicle LKAS system outperformed the random generator in terms of the F_1 fitness evaluated in simulator of the solutions produced. The RL agent generated test scenarios that were 34.5% and 88.5% fitter for the autonomous ant robot and autonomous vehicle, respectively. The random generator produced test scenarios with higher diversity, by 4.46% and 14.3% for the robot and vehicle, respectively. The RL test generator for the autonomous vehicle still achieved a high level of diversity (0.839), meaning that the test scenarios were at least 83.9% different from each other. The overhead of the RL generator in terms of generation time is insignificant.

4.4.4 Threats to Validity

Internal validity. To minimize the threats to internal validity, relating to experimental errors and biases, whenever available, we used standardized frameworks for development and evaluation. We implemented the evolutionary search algorithms based on a popular Python-based Pymoo framework [179]. We used the PPO RL algorithm implementation provided by the Stable-baselines framework [163]. For the autonomous robot problem, we used an open-source and standard RL Ant-maze environment based on Gymnasium library [189], [191]. To evaluate the scenarios for the LKAS case study, we utilized a standardized test pipeline from the SBST 2022 workshop tool competition [155].

To minimize the threats to internal validity stemming from selected parameter values, we conducted extensive experimentation with various hyperparameters for our approach. This included exploring different values for ρ percentage, as well as employing multiple Multi-Objective Evolutionary Algorithms (MOEAs). Such evaluation allowed minimizing the potential for bias of specific parameter choices on our results. When comparing RIGAA to other baselines, we chose the configuration of RIGAA where $\rho = 0.4$, that showed the best performance in terms of failures and sparseness in our previous experiments. Finally, we would like to mention the threat related to the execution of scenarios in a simulated environment. While simulation-based testing is a widely accepted strategy for autonomous CPS testing [38], it's important to acknowledge that some failures identified in simulations may not accurately reflect real-world conditions. To bridge this gap, we have chosen to utilize realistic simulators such as BeamNG [148] and the widely used simulator Mujoco [188].

Conclusion validity. Conclusion validity is related to random variations and inappropriate use of statistics. To mitigate it, we followed the guidelines in [192] for search-based algorithm evaluation. We repeat all evaluations a statistically significant number of times, i.e. at least 10 when using the simulator and at least 30 otherwise. For all results, we report the non-parametric Mann-Whitney U test with a significance level of $\alpha = 0.05$, as well as the effect size measure in terms of Cliff's delta.

Construct validity. Construct validity is related to the degree to which an evaluation measures what it claims. In our experiments, we used two evaluation metrics: the test scenario fitness as well as test scenario diversity. We evaluated the fitness proportionally to the safety requirement violations. For the autonomous ant robot, we calculated the percentage of the target trajectory completed and for the autonomous vehicle, the maximum deviation from the road lane. For the evaluation of diversity, we utilized the Jaccard distance, which compares test scenarios' similarity based on their input representation. This metric provides normalized diversity values, which is useful for comparison on different testing problems. However,

it only evaluates the differences in the inputs given to the system, and not the differences in the output behavior of the system. In the future, we plan to also include metrics for evaluating the diversity in the output behavior coverage by the test scenarios. Finally, we conclude that RIGAA approach outperforms the available baselines in terms of the number of revealed failures. We acknowledge that this finding is true, provided that RIGAA is allocated a one time training budget to develop the RL agent.

External validity. External validity relates to the generalizability of our results. We demonstrated how our framework can be applied to generate environments for two different autonomous robotic systems. However, we only considered a limited number of test subjects and limited levels of environment complexity. Therefore, additional experiments should be conducted with different robotic systems and environments with higher complexity for better generalizability evaluation of RIGAA. We have also experimented with two different multi-objective evolutionary algorithms (MOEA) to be combined with a pre-trained RL agent. Both of them have shown positive results, however, we need to conduct experiments with a bigger variety of search techniques, such as evolution strategies and particle swarm optimization [193], to confirm that RL-based initialization is beneficial for different search algorithms.

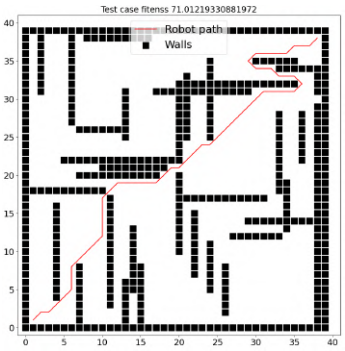
4.4.5 Data Availability

The replication package of our experiments, including the implementation of the search algorithms and the instructions for installation of the simulators we used, is available at Zenodo repository [194]. A non-random initial population can direct the search into particular regions of the search space that contain good solutions [40]. This is of special interest to search-based testing of autonomous robotic systems, where the cost of solution evaluation is high. By reducing the time spent on evaluating non-promising solutions, more failures can be potentially revealed in less amount of time. Our search-based approach, RIGAA, leverages a pre-trained RL agent to insert some proportion of high-quality test scenarios in the initial population. Preliminary results have shown that RL informed initialization allows the search algorithm to find better test scenarios within the same computational budget, in contrast to a configuration based on random initialization. In this section we discuss in a more detail the advantages and disadvantages of RL-based initialization as well as the promising directions for expanding our work.

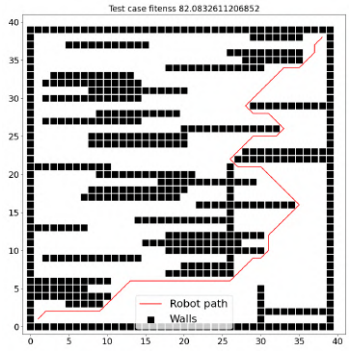
4.5 Discussion

4.5.1 Discovering More challenging Test Scenarios with RL-based Initialization

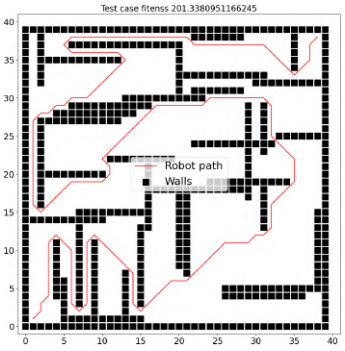
In RIGAA, instead of manually designing domain knowledge-based initialization heuristics or reusing pre-existing scenarios, we rely on the RL agent to automatically infer the domain knowledge rules via continuous interactions with the provided environment. The domain knowledge can be obtained by both interactions with the actual simulator environment or the environment with surrogate rewards. We opted for the latter in order to reduce the computational cost of training the agent. Our experiments show that as long as the RL agent produces test scenarios with higher fitness, than the random generator, it can be useful for initialization of the evolutionary search. We surmise that incorporating individuals with initial problem knowledge into the evolutionary search process can lead to the discovery of new types of challenging test scenarios that a purely stochastic search process may fail to discover within a reasonable computational budget.



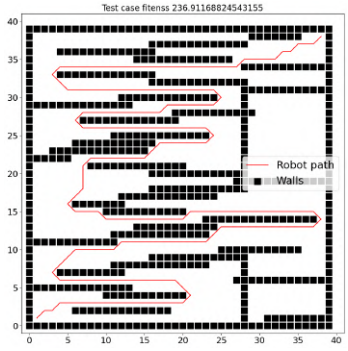
(a) Randomly produced scenario



(b) RL produced scenario



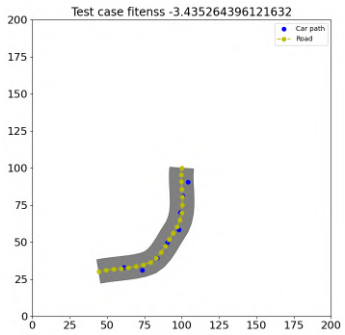
(c) NSGA-II produced scenario



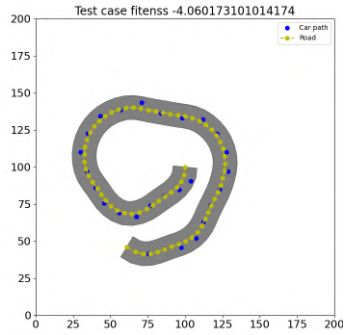
(d) RIGAA produced scenario

Figure 4.25 Comparing test scenarios for an autonomous robot

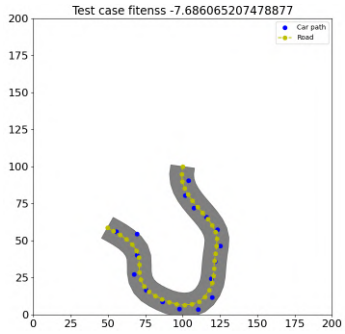
As an example, in Figure 4.25 we show examples of the generated mazes for an autonomous



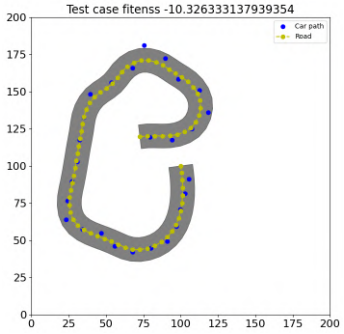
(a) Randomly produced scenario



(b) RL produced scenario



(c) NSGA-II produced scenario



(d) RIGAA produced scenario

Figure 4.26 Comparing test scenarios for an autonomous vehicle

robot produced randomly, with an RL agent, with NSGA-II and with RIGAA. In Figure 4.25b and Figure 4.25d we can observe a maze pattern with mostly horizontal walls, produced with an RL agent and RIGAA respectively. Firstly, this pattern was introduced into the initial population by a trained RL agent, then it was evolved with an evolutionary search, producing a maze with the highest fitness observed in our experiments. It is different from a typical maze pattern that was discovered by random search and NSGA-II, shown in Figure 4.25a and Figure 4.25c that has arbitrary choices of vertical and horizontal walls.

In Figure 4.26 we show the corresponding examples of road topologies produced for testing autonomous vehicles. In Figure 4.26d we can see a curved road topology pattern, produced by an RL agent. After being evolved by evolutionary search, more challenging road topology was produced (Figure 4.26d), forcing the vehicle to go further from the road lane center, than in NSGA-II discovered test scenarios.

The examples presented above illustrate that RIGAA can be useful for discovering new types and patterns of test scenarios, potentially revealing previously unseen system failures.

4.5.2 Challenges of Using RL Agents for Test Scenario Generation

In RIGAA we use a pre-trained RL agent to generate some percentage of the initial population. Below, we outline some of the challenges related to training an RL agent for test scenario generation.

Sample inefficiency. RL agents typically require a large number of interactions with the environment to learn effective policies. The process of training the RL agent may be computationally expensive and time-consuming, especially if the RL agent needs to explore a complex and high-dimensional action space. For an autonomous robot, the action space to explore was multidimensional and of the size $2x7x37$. The average time to compute the reward value was 0.1 s. We trained the agent for a total of 500 000 steps, taking around 13.5 hours on average. For an autonomous vehicle, the action space was of the size $3x25x35$, with the average time to compute the reward of 0.01 s. We trained the agent for a total of 2 500 000 steps, taking around 12 hours on average. When the problem action space is high-dimensional, we recommend the use of an inexpensive reward function, as the computation of the rewards based on the system behavior in the simulator becomes prohibitively expensive. In our case studies, test scenario evaluation in Mujoco simulator (autonomous robot) could take from 20 to 50 s, and in the BeamNg simulator (autonomous vehicle) from 30 to 60 s.

Reward shaping and hyperparameter fine-tuning. RL algorithms typically have several hyperparameters as well as a reward function that require careful tuning to achieve good performance. In the context of testing, we are interested in producing not only challenging but also diverse test scenarios. One of the important steps is to choose the reward function encouraging challenging scenarios as well as the hyperparameters that provide a good exploration-exploitation trade-off. Otherwise, the trained RL agent might learn to generate only a specific type of test scenario, lacking diversity. In our experiments, we gave additional rewards to the agent for selecting novel elements for the test scenarios. Additionally, we fine-tuned the entropy coefficient hyperparameter, that balances exploration and exploitation of the agent.

Limitations in application domains. To be able to use RL, the test scenario problem should be first formulated as an MDP. If it is not possible in a given problem, RIGAA may not be applicable. We hypothesize that the majority of test scenario generation problems for autonomous robotic systems can be formulated as an MDP.

Surrogate reward design. In some test generation problems, it can be challenging to formulate the domain knowledge rules and apply them to design the surrogate rewards, that serve as a proxy for the rewards obtained from executing the simulator-based model of the

system. One solution is to directly utilize the feedback from the simulator-based model as a reward for training the RL agent. In this case, the action space should be minimized, to be easier for the agent to explore.

Overall, training an RL agent requires additional effort for the creation of the training environment, as well as the computational resources to train the agent. However, once the agent is trained to produce challenging and diverse scenarios, it can be re-used as a generator of test scenarios, with minimal costs of generation. In our experiments, the efficiency of the RL generator was comparable with that of the random generator. Moreover, it can help uncover new types of failures, not previously found with randomly initialized algorithms. In this case, the performance on system falsification may take priority over the algorithm cost and complexity.

4.6 Chapter Summary

In this chapter, we proposed RIGAA, a search-based approach that improves the effectiveness of search-based closed-loop testing of autonomous robotic systems. By utilizing a pre-trained RL agent, RIGAA incorporates operational knowledge into the test generation process, enabling effective exploration of safety requirement violating scenarios. At the same time, RIGAA reduces the evaluation time spent on irrelevant tests. We evaluated RIGAA on two systems: an RL-controlled Ant robot and a vehicle lane keeping assist system. Our results demonstrate that RIGAA outperforms the search algorithms with random initialization. Notably, RIGAA outperformed NSGA-II by producing 37.4% more failures for the autonomous ant robot in the Mujoco simulator and 34.2 % more failures for the autonomous vehicle in the BeamNG simulator. RIGAA also discovered 15.5 % more failures than CRAG, 23.5 % more failures than AmbieGen and 82.1 % more failures than the Frenetic state-of-the-art LKAS testing tools, given a two-hour evaluation budget.

Combining a trained RL agent with evolutionary search consistently resulted in the discovery of more failures than using the RL agent alone. Moreover, our manual qualitative analysis confirmed that this integration uncovered additional failure patterns that were not revealed by evolutionary search.

CHAPTER 5 REPRESENTATION IMPROVEMENT IN LATENT SPACE FOR SEARCH-BASED TESTING OF AUTONOMOUS ROBOTIC SYSTEMS

5.1 Problem Context

The architectures of modern autonomous robotic systems (ARS), such as self-driving cars, are highly complex, integrating hardware, software, and machine learning components that interact in dynamic and often unpredictable ways. Consequently, ensuring the robustness of ARS software is essential to enable these systems to withstand the numerous challenges present in unstructured real-world environments.

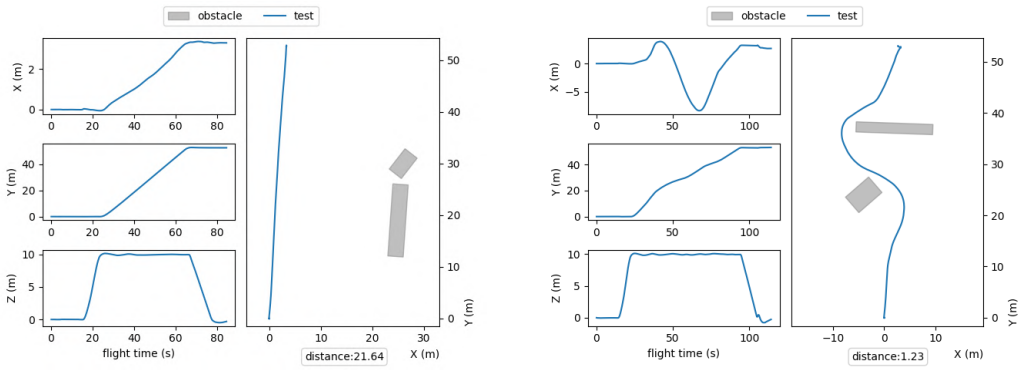
Simulation-based testing is a widely adopted method for evaluating ARS performance, where system models are assessed under various scenarios within a virtual environment. However, these simulations are computationally expensive, since each run can take from tens of seconds to several minutes. Given the limited testing time budget, selecting the most promising simulations is critical. Random test selection typically generates many irrelevant or non-challenging scenarios, thereby reducing efficiency. To overcome this limitation, several search-based approaches have been proposed for more efficient test generation [24, 95, 97, 118]. The effectiveness of these approaches is heavily influenced by the choice of representation, which should facilitate the discovery of challenging test scenarios while promoting a diverse set of tests [125].

Recent techniques in autonomous systems testing have specifically targeted the need for diverse scenario representation. For example, the Doppelganger tool [195] for testing autonomous driving systems specifies test cases by detailing the positions and trajectories of autonomous vehicles (AVs) and pedestrians, as well as dynamic traffic signal configurations. In Doppelganger, each AV is represented by a set of 2D points outlining its expected path along with a movement start timestep, while each pedestrian is characterized by a set of 2D points, a movement start time, and a speed. Traffic control configurations are defined by five parameters that specify the signal color at each moment of the simulation. Although this expressive representation enables the specification of a vast number of test cases, it also leads to increased complexity in the scenario search space. Given the non-negligible time required for each simulation, efficiently exploring this complex space—even with search-based techniques—remains a significant challenge.

Concurrently, recent work in the evolutionary search community [129, 196, 197] has demonstrated the benefits of leveraging the latent space of generative models, such as generative

adversarial networks (GANs) [198] and variational autoencoders (VAEs) [199], for optimization tasks. The latent space offers a smoother, more continuous search landscape and encodes a bias toward high-performing solutions derived from the data. Intuitively, these approaches transform a difficult-to-search genotype space into a more structured and navigable latent space.

To the best of our knowledge, no approach in the search-based testing or ARS literature has yet explored mapping the genotype search space to a latent space to improve representation. To address this gap, we propose Representation Improvement in Latent Space for Search Based Testing of Autonomous Robotic Systems (RILaST), a framework that leverages evolutionary search in conjunction with variational autoencoders to optimize the search over a latent space populated with challenging and diverse test scenarios.



(a) Test with obstacles far from the UAV path

(b) Test with obstacles close to the UAV path

Figure 5.1 Examples of non-failing and failing test scenarios for the UAV system

To illustrate the potential benefits of learning a latent space for test generation, consider an unmanned aerial vehicle equipped with an obstacle avoidance system. The UAV is required to complete a mission by flying straight for 50 meters while maintaining a distance of at least 1.5 meters from any obstacle. As shown in Figure 5.1a, a test scenario with obstacles located far from the UAV's flight path is less likely to induce a failure, making it less valuable for testing. Conversely, applying a simple heuristic—such as positioning obstacles closer to the UAV's path—can significantly increase the difficulty of the test scenario, as depicted in Figure 5.1b, where the UAV fails to meet the safety requirement. However, explicitly incorporating this heuristic as an additional constraint may lead to a more discontinuous search space, complicating the search process [200]. This motivates our approach of optimizing test generation within the latent space, where an improved representation can be learned from

data containing diverse, failure-revealing tests. Our experiments on two use cases demonstrate that this improved representation enables the search algorithm to navigate the space more efficiently, uncovering 3–4.5 times more diverse failure-revealing scenarios compared to searches in the original space.

In this chapter, we make the following contributions:

1. RILaST, a search-based framework that enhances representation for search-based testing by learning it in the latent space of a variational autoencoder;
2. An extensive evaluation of RILaST and comparison to baselines, including state-of-the art tools and alternative implementations of the search algorithm used in RILaST;
3. A publicly available replication package with the implementation of RILaST [201].

5.2 Problem Formulation

In this section, we formulate the problem of generating test scenarios for autonomous robotic systems as an optimization problem. We then provide a detailed description of this problem in the context of two case studies: autonomous lane keeping and autonomous navigation.

5.2.1 Test Scenario Generation Problem

We begin by formally defining the problem of test scenario generation as an optimization problem. This formulation includes constraints to ensure the generation of valid test scenarios and objectives aimed at falsifying the requirements of the system under test.

Given a simulator Sim, an autonomous system under test (SUT) operates in a simulated environment E defined within Sim. A simulated environment is defined in terms of temporal parameters P, static and dynamic objects O_s and O_d. Examples of parameters P include the illumination level, time of the day, weather conditions, etc. The value of these parameters may evolve during the scenario execution or remain static. Static and dynamic objects are defined via parameters P_{os} and P_{od}, respectively, describing their location, orientation, shape, etc. For static objects, the values of their parameters P_{os} remain constant during the simulation execution. For the dynamic objects, the values of P_{od} evolve at each timestamp t of running the simulation. Constraints C can be imposed on the values describing O_d and O_s. The SUT is defined by its state, S, that evolves at each timestamp t. In a given environment, the SUT has a mission M to complete. Examples of missions include following a specific trajectory or navigating from a *start* location to a *goal* location. We define the

environment E with the SUT assigned a specific mission M as a test scenario TS. Upon executing TS in a simulator Sim, the observed SUT states S_t are recorded in a system state trace $T = (S_0, t_0), (S_1, t_1), ..., (S_N, t_N)$, where N is the number of simulation steps. At all times of operation, SUT is expected to satisfy certain requirements $R_i \in \mathbf{R}$, where $\mathbf{R}$ is the full set of requirements. Each requirement R_i is specified via a corresponding signal temporal logic (STL) formula φ_i. STL is a commonly used formalism for describing the behavior of cyber-physical systems [202]. In our study, we consider the STL requirements in the following form:

$$\varphi_i = \Box(s_i(t) > \epsilon_i) \tag{5.1}$$

which states that at each time step, i.e., the value of the system state signal $s_i(t) \in T$ should "always" be greater than a certain threshold ϵ. A system state signal s_i represents specific values derived from the system state S that are used to assess whether the system meets a particular requirement R_i. For example, in the simulation of an autonomous vehicle, relevant state values in S may include the deviation from the lane and the minimum distance to pedestrians. The state signal s_1, associated with the "staying within the lane" requirement, would correspond to the vehicle's deviation from the lane. Similarly, the state signal s_2, associated with the "keeping a safe distance from pedestrians" requirement, would represent the minimum distance to pedestrians. We employ the robustness metric ρ [203] to evaluate the extent to which the property φ_i is satisfied. Robustness ρ_i of satisfying requirement R_i typically corresponds to the difference between the minimum $s_i(t)$ observed and the established ϵ_i threshold:

$$\rho_i = \min_t s_i(t) - \epsilon_i \tag{5.2}$$

For instance, in the case of an AV avoiding collisions with pedestrians, ρ_i can correspond to the difference between the minimum distance observed and the established safe distance threshold. If the AV maintains a safe distance, the value of ρ_i will be higher. When the SUT has multiple requirements R_i, we use the global robustness metric ρ_{safe}, to measure the extent to which the simulation trace T falsifies the established requirements:

$$\rho_{safe}(T) = \min_{s_i \in T} \rho_i(s_i) \tag{5.3}$$

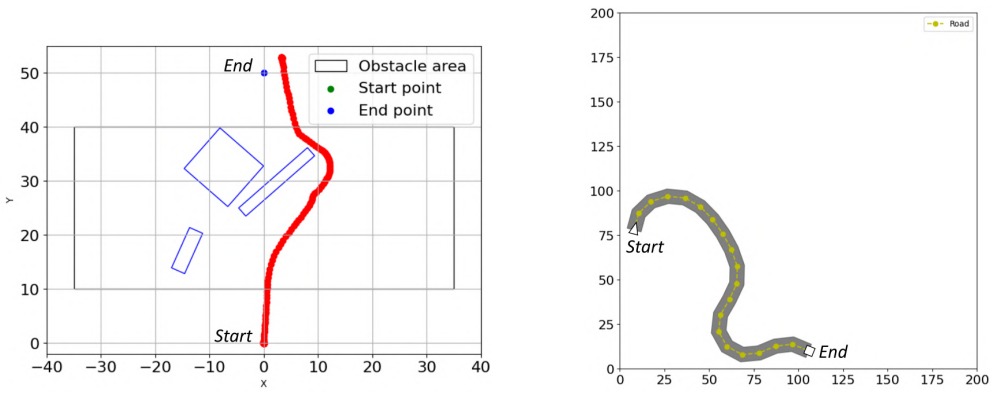
Robustness ρ_{safe} corresponds to the lowest robustness ρ_i observed in a trace T. The goal of test generation is to manipulate the parameters of TS in such a way that system traces $\hat{T}$

with the lowest possible values of ρ_{safe} are produced:

$$\hat{T} = \arg \min_{T \in \mathcal{L}(\Sigma)} \rho_{safe}(T) \tag{5.4}$$

were $\mathcal{L}(\Sigma)$ represents the set of all possible system traces. In order to solve this non-linear, non-convex optimization problem, stochastic search optimization methods can be applied [78]. In the context of testing ARS this is a challenging task. First, the possible search space for test scenarios is vast. Second, the produced traces $\hat{T}$ should be diverse. Third, running a simulator for collecting traces T is often time-consuming due to the intensive computations required for scene rendering. In our work, we aim to facilitate the search by improving the representation of the problem. In the following subsection, we describe the test generation problems for each of the two studied use cases.

5.2.2 Obstacle Positioning



(a) Test example for UAV

(b) Test example for UAV

Figure 5.2 Examples of the considered test scenarios

The first test generating problem we consider is the placement of rectangular shaped obstacles in an enclosed environment. This test generation problem was previously proposed by Khatiri et al. [204] and used for the SBFT-2024 CPS-UAV competition [85]. Such tests can be used to evaluate the performance of the navigation algorithm of an autonomous drone. Here, the temporal parameters P such as weather and illumination correspond to normal conditions, with a sunny day illumination level and remain fixed. In this environment, there are no dynamic objects O_d; i.e., all objects are assumed to be static. The number of static objects O_s varies from 1 to 3. Objects O_s are rectangular obstacles, each described with 6 parameters, such as: x, y coordinates, width (w), length (l), height(h), and orientation (r). The following

constraints C are imposed on static objects: (1) objects cannot intersect, and (2) objects cannot completely block the path for the SUT. The mission of the SUT is to complete the path from the start to the goal location. The SUT has only one requirement, R, which is to remain at a distance of more than 1.5 m from the obstacles during the flight. Here, the system state signal that we record during the simulation execution is the minimal distance to any of the obstacles, which should be bigger than the threshold $\epsilon = 1.5$. The goal of the test generation algorithm is to find such combinations of obstacles positions, shapes, and orientations that force the SUT to violate the requirement. An example of one test is illustrated in Figure 5.2a. It consists of 3 obstacles with the drone trajectory shown in red. The mission consists of flying straight from the start to the goal location. The average execution time for a single test scenario ranges from 4 to 5 minutes.

5.2.3 Road Topology Generation

In the second problem, the goal is to create road topologies that cause the lane keeping assist system (LKAS) of an autonomous vehicle to fail. This test generation problem was used at SBFT CPS testing competition 2023 [181]. Similarly to the previous problem, the temporal parameters, such as weather and illumination, remain fixed and correspond to a sunny day. No dynamic objects O_d are present in the environment. One static object O_s is present, which is the road topology with two lanes. The road topology is defined with 2D points, that are then approximated with cubic splines to obtain a smooth trajectory. The number of road points can be variable, with at minimum two points, so that they can be approximated with splines. The following constraints C are imposed on the road topology: (1) the road has to remain within the map borders, (2) the road cannot be too sharp, and (3) the road cannot self-intersect. An example of a road topology containing 17 2D control points is shown in Figure 5.2b. The mission M of the SUT is to follow the trajectory of the road topology from the start to the end location. These locations are given by the first and last points defining the road topology. The main requirement R for the SUT is to stay within the lane boundaries, with a threshold of $p = 0.85$ (as defined at the SBST 2021 competition [160]), meaning that the vehicle is considered to be out of the lane if more than 85% of it lies outside the lane boundaries. As the system state signal, we record the percentage of the vehicle that stays within the lane, which should be bigger than the threshold $\epsilon = 0.15$. The test generation algorithm should produce such road topologies that force the SUT to violate the requirements and go out of the lane bounds. One test scenario simulation takes from 0.5 to 1 minute on average.

5.3 Approach

RILaST approach aims at improving the original test scenario representation. The new test representation obtained with RILaST should make it easier for the search algorithm to navigate the search space and converge towards failure revealing test cases. The RILaST approach includes three key steps: (1) collecting a dataset of test scenarios, (2) training a variational autoencoder (VAE) model, and (3) executing an evolutionary search within the latent space of the autoencoder. These steps are illustrated in Figure 5.3. We present the detailed explanation of each of the steps below.

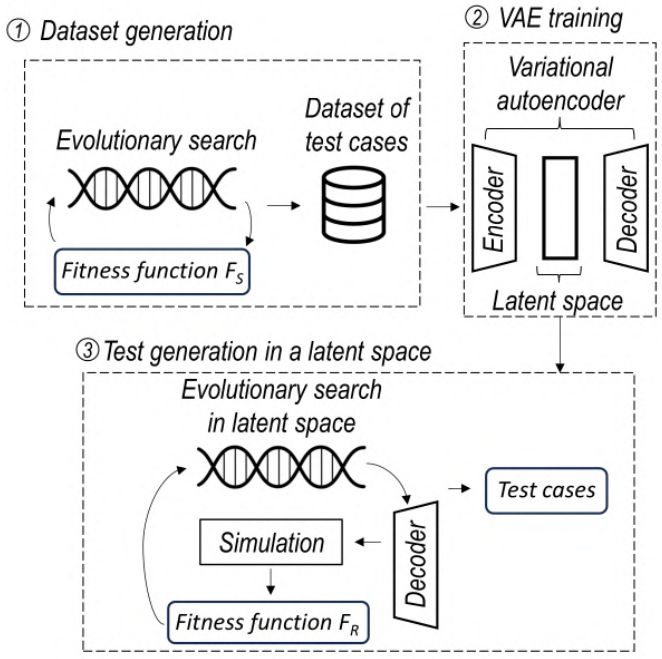


Figure 5.3 RILaST approach diagram

5.3.1 Dataset Generation

An important step in VAE training is the collection of the dataset D. Inherently, a VAE learns to generate samples from a similar distribution as the training dataset. Therefore, it is important to ensure the dataset of test scenarios is diverse, but at the same time contains relevant and challenging test cases. Previous approaches in evolutionary search literature propose running a genetic algorithm [128] or illumination search [125] to produce an initial dataset for VAE training. In the test generation context, we propose to first run a genetic algorithm guided by a simplified objective function several times, adding the population from

the last generation to the dataset. We provide a more detailed description of the simplified function in the next subsection. We chose genetic algorithms as they have previously proven effective in test generation [185]. As another baseline, we also consider generating an entire dataset randomly, where each test is produced by uniformly sampling the allowed parameter ranges. Running the search algorithm multiple times allows exploring a bigger search space, ensuring the diversity of the solutions in the dataset. Moreover, we add diversity preserving mechanisms to the search itself, to ensure that the solutions in the last generation are diverse. To estimate the number of runs N of the genetic algorithm we divide the final dataset size N_{data} by the population size, i.e., $N = \frac{N_{data}}{P}$ For both use cases, we collected datasets of 10 000 entries for running a genetic algorithm with a population of 200 for 50 generations 50 times.

Simplified function design

To guide the generation, we propose using a simplified fitness function F_s that is based on some knowledge of what constitutes a challenging test scenario. The function F_s should be inexpensive to evaluate to be able to generate the dataset in a reasonable amount of time. In Figure 5.4 we provide some guidance in choosing the function. Firstly, the knowledge can be contained in a simplified, i.e., surrogate model of the system. Such model is much less expensive to execute, and it provides approximate, but accurate enough results. Such models can be derived using traditional system identification methods [71] or supervised machine learning algorithms [96]. For instance, Haq et al. [24] developed a surrogate model for the Pylot [164] system in the CARLA simulator. Birchler et al. [96] introduced SDC-scissor, which is a supervised learning-based model predicting the output of the scenarios for LKAS testing. Moreover, some systems, such as mobile robots, have kinematic models available [165], which can also serve to approximate their behavior. The surrogate model's predictions on the test scenario outcome can then be utilized as a fitness function during the collection of the dataset.

Next, if the surrogate model is not available, we suggest some heuristics from autonomous system testing literature. For instance, when designing test scenarios with dynamic obstacles, such as vehicles or pedestrians, Babikian et al. [205] propose to prioritize test cases where the paths of the SUT and the obstacles intersect. If static obstacles are present in the scenario, Humeniuk et al. [157] propose to prioritize test scenarios where the path planning algorithms, such as RRT or A* return a longer path. When the SUT is required to follow a certain trajectory, Zohdinasab [97] showed in their experiments that complex trajectories, involving turns of a high level of curvature, are more likely to cause failures. Finally, if none of

these cases apply, test scenarios can be prioritized based on their novelty and the satisfaction of constraints. Specifically, we propose to run the novelty search algorithm, as proposed by Lehman et al. [206], where the objective function is defined as the average diversity between a given solution and its n nearest neighbours.

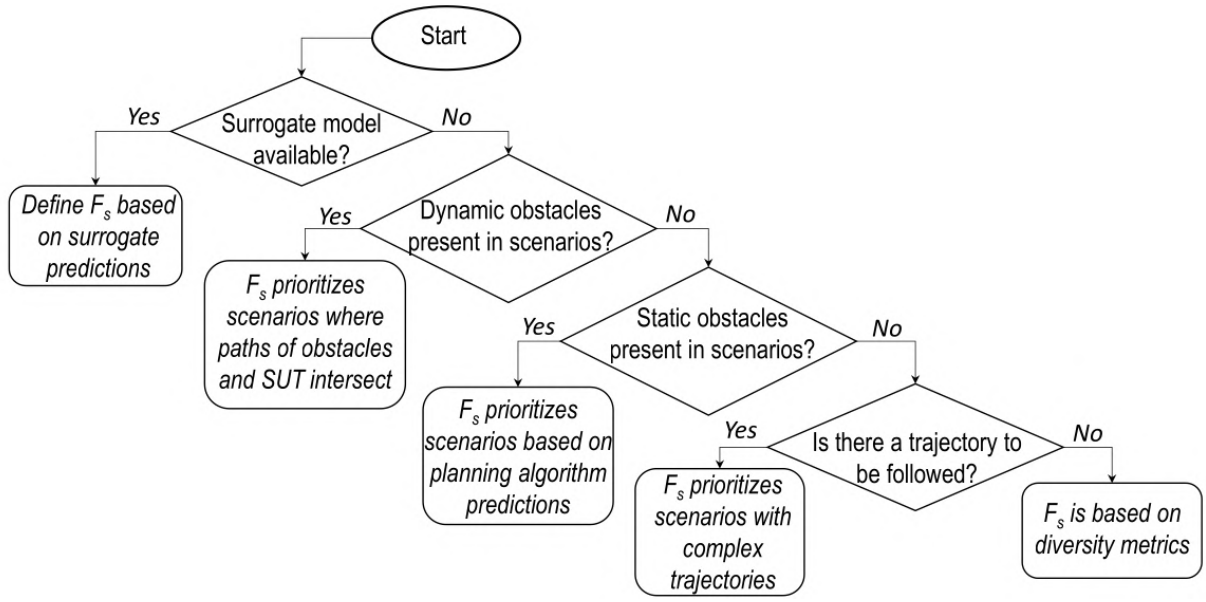


Figure 5.4 Simplified fitness function design

Below we provide more details on the genetic algorithm configuration for the generation of the dataset. When there are multiple falsification objectives, algorithms for multi-objective optimization such as NSGA-II [47] or MOEAD [207] can be used. In our case, we leveraged a GA with a single optimization objective [40].

Genetic algorithm design

In the following, we present the implementation details of the genetic algorithm. We refer to this GA configuration as GA_D, in other words GA for dataset collection.

Individual representation and sampling. We represent the individuals, or chromosomes, as one-dimensional arrays of size M. It is a standard way of representing individuals in evolutionary computation [40], i.e., arrays of real values, and it simplifies the use of existing search operators, such as crossover and mutations. For the obstacle placement use case (UC1), each test case is described by a 19-dimensional array. The first element corresponds to the number n of box-shaped obstacles in the scene. The remaining 18 elements describe the obstacles that are present in the scene (up to 3). Each obstacle is described by 6 elements: x and y

coordinates of the obstacle center position, obstacle length l, width w, height h in meters, and rotation r in degrees.

Table 5.1 Allowed ranges for the parameters for the studied use cases

n	x	y	l	w	h	r	gb	rb
$1-4$	$-40-30$	$10-40$	$2-20$	$2-20$	$15-25$	$0-90$	$-0.07-0.07$	0.05

For the second use case (UC2), to define the road topology points we leveraged a curvature-based representation introduced by Castellano et al. [120], where a road topology is described by a sequence of N curvature values (kappa) κ_0, κ_1, ..., κ_{N-1}. These values can then be mapped to 2D points with a simple transformation. Based on the experimental results, we chose to have a fixed size of a kappa vector of 17. Having a fixed-size vectors also simplifies the VAE training. When generating the initial vectors of kappa values, the value of κ_i is obtained from a range defined by a global bound and a relative bound. The global bound (gb_l, gb_u) defines the minimum and the maximum value that $kappa_i$ can take, while the relative bound rb defines the difference between k_i and its previous value. The values of these bounds are indicated in Table 8.1. In both use cases, for the initial population, we generate N_{pop} vectors randomly by sampling from the allowed value ranges, which are listed in Table 8.1. N_{pop} corresponds to the population size.

Fitness function. To guide the search for dataset collection, we used a simplified F_s fitness function. In the obstacle placement problem, the goal is to find a configuration of static objects that will reveal a SUT failure. Based on Figure 5.4 we select F_s to be based on the RRT* planning algorithm [208]. Intuitively, the longer the path produced by the planner, the more deviations from the shortest path the UAV will have to make, increasing the risk of its failure.

For the topology generation, we used the maximum road curvature value as the objective function. Intuitively, the bigger the curvature angle is present in the road topology, the more challenging it should be for the driving agent. When the maximum road curvature exceeded the limits, the test cases are given a low fitness value.

Crossover operator. We are using a one-point crossover operator, which is one of the commonly-used operators [209]. Given a pair of parents, the crossover point is randomly determined. At this point, genes (components of the chromosomes) are exchanged. This operator is applied with probability p_{cross}. Following the generation of the offspring, test scenario constraint C feasibility validation should be performed. If the constraint is violated, a low fitness value is assigned to the individual.

Mutation operator. For both use cases, we use domain-specific mutation operators. After

applying the crossover operator, one of the presented operators is applied with a probability p_{mut}. For the obstacle placement problem, three operators are used: random modification to one obstacle, obstacle addition, or obstacle removal. During obstacle modification, one of the obstacles was randomly selected and its parameters, such as x, y, w, h, l, r were assigned random values from the available ranges. During obstacle addition, one obstacle with randomly sampled parameters is added, if the maximum number of obstacles in the scene is not exceeded. During the obstacle removal, one obstacle is randomly selected and removed, ensuring that at least 1 obstacle is left in the scene.

For the road topology generation, we leveraged the mutation operators proposed by the FreneticLib framework [210]. These operators include: increasing kappa values from 10 % to 20 % (increase operator), changing the values of n randomly selected kappas (change operator), reversing the order of kappa values (reverse operator), changing the sign of kappa values (sign change operator). In 'increase' operator, the value of the increase is uniformly sampled from 10 % and 20 % and applied to all the kappa values. In 'change' operator, we randomly select up to 5 kappa values and randomly change their values within global bounds *gb*. In the reverse operator, we reverse the order of all the kappa values. In 'sign' change operator, the signs of all the kappa values are changed to the opposite one. Having a diverse set of mutations helps generate a set of diverse test scenarios.

Duplicate removal. At each iteration, we remove the individuals which have a small deviation between them. We estimate the deviation as the cosine distance D_c [211] and compare it to a prefixed minimum threshold, D_{cth}. If the D_c between the two individuals is less than D_{cth}, one of them is removed.

The hyperparameters used for this GA configuration are specified in Table 5.2.

Table 5.2 Hyperparameters of GA configuration for the dataset collection

Parameter	Description	Values
N_{pop}	Population size	200
p_{cross}	Crossover rate	0.9
η_c	Initial crossover η	3
p_{mut}	Mutation probability	0.4
η_m	Initial mutation η	3
D_{cth}	Duplicate threshold	0.025
N_{off}	Number of offspring	100

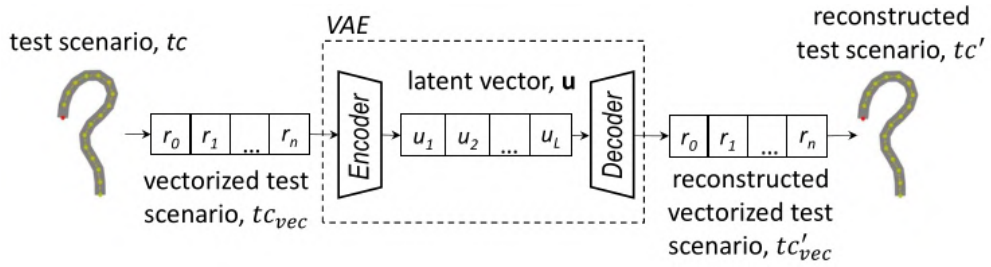


Figure 5.5 Test case encoding and decoding with VAE

5.3.2 VAE Training

The primary objective of training the VAE is to learn a mapping function between a given test case and its representation in a latent space. The process of transforming a test to the latent space and then back into the original space is shown in Figure 5.5. Test scenarios tc are given as the input to the simulation environment. In the figure, the test corresponds to a road topology, represented as a sequence of 2D points. As presented earlier, in order to perform optimization, test scenarios should be transformed to a simpler genotype representation, such as 1D array of size M, which we refer to as vectorized test scenario tc_{vec}. Mapping from tc to tc_{vec} typically is done with a deterministic function. Traditional search-based approaches perform the search in the vectorized test scenario space. Our goal is to use the obtained dataset D and the VAE to learn a better representation to perform the search in. We refer to this representation as the latent vector u. The search space of latent vectors is referred to as latent space. To map the tc_{vec} to $\mathbf{u}$, the encoder part M_E of the VAE is used. At the output of the encoder we get the latent vector $\mathbf{u}$ of size L. The values of this vector follow a normal distribution with a mean of 0 and a standard deviation of 1. The size of L is typically smaller or equal to M. Reducing the size of L can help reduce the search space size, but at the same time it can increase the reconstruction error of the original input tc_{vec}. Therefore, we chose to have the size of L equal to the input space size M. To map the vector from the latent space u to the original vectorized test scenario space, the decoder module M_D is used. Passing $\mathbf{u}$ through the decoder, we obtain the reconstructed vectorized test scenario $tc^{'}_{vec}$. With a trained VAE, we expect the mean squared error (MSE), i.e., reconstruction loss between the tc_{vec} and $tc^{'}_{vec}$ to be minimized. Achieving low reconstruction loss is important to ensure that the VAE learns a meaningful representation. A poorly learned latent representation results in an inconsistent mapping between the latent variable and the actual variable, complicating the evolution process [127]. Having a well reconstructed $tc^{'}_{vec}$ is also important for obtaining meaningful and valid reconstructed test scenarios $tc^{'}$ in the end. Below, we provide more details on our VAE training setup.

VAE is a neural network, consisting of two main parts: encoder and decoder, with symmetric architecture. During the training phase, both encoder and decoder parts are used. During the search in the latent space only the decoder is required. We trained the VAE with a set-up recommended by Bentley et al. [127]. The encoder consists of two hidden layers with sizes 128 and 64, respectively. The decoder has the same architecture but in reverse order. The VAE was trained for 1000 epochs using the Adam optimizer with a learning rate of 0.001 and a batch size of 512. To ensure smoother activation values during training, we normalized the dataset inputs to the range $[-1, 1]$ and applied the tanh function at the output layer. The VAE learns to represent the data within a latent space $\mathbf{z}$, where latent vectors $z_i \in \mathbf{z}$. To minimize the final loss, we set the dimensionality of the latent vectors equal to the input size. We train a separate VAE for each use case, which we refer to as VAE_{uc1} and VAE_{uc2} for the first and second use cases, respectively.

5.3.3 Test Generation in the Latent Space

Having an improved representation in a latent space of the VAE, we can perform the search over this space. We leverage a genetic algorithm with a simulator-based objective function. We also use the decoder part of the VAE M_D to map the test specification in the latent space to the original space. Below, we provide a more detailed description of the GA we use to perform the search. We refer to this GA configuration as GA_L, in other words, GA for optimization in a latent space.

Individual representation and sampling. We represent the individuals, or chromosomes, as one-dimensional arrays of the size of the latent space L. In our implementation, the latent space size is equal to the original representation size. For the obstacle placement use case, each test case is described by a 19-dimensional latent vector, while for the topology generating use case, a test case is described by a 17-D vector. The key feature of the latent space is that each dimension of the latent vector $\mathbf{u}$ is designed to follow a normal distribution with a mean of 0 and a standard deviation of 1, i.e., $\mathbf{u} \sim \mathcal{N}(0, I)$, where I is the identity matrix. Therefore, to generate the random population, we simply sample L values from a normal distribution $\mathcal{N}(0, 1)$ to create each of the N_{pop} latent vectors $\mathbf{u}$. Here, N_{pop} corresponds to the population size.

Fitness function. To guide the search in the latent space, we use the simulator-based fitness function F_{sim}. We evaluate it based on Eq. (5.2), which computes the deviation between the recorded system execution trace T, i.e., the observed behavior, and the given requirement, i.e., the expected behavior. The system execution trace is obtained after running the given test scenario in the simulation environment. For UC1, as the system trace T, we recorded the

closest distance from the SUT to an obstacle at each timestamp. The SUT requirement R was to complete the flight mission without approaching the obstacle too closely, i.e., flying at a distance of more than 1.5 meters. As the fitness function F_{sim}, we calculated the minimal distance to the object observed. For UC2, we recorded the percentage of the SUT going out of the road bounds. The requirement was to avoid going out of the road bounds. As the fitness function, we calculated the maximum percentage of the SUT observed to be out of the lane.

Crossover operator. The representation in the latent space, i.e., a vector of L independent real values, aligns with the standard representation used in the evolutionary search domain [40]. Therefore, all real-valued search operators developed by the evolutionary search community can be applied. We selected the self-adaptive simulated binary crossover [212] as one of the most advanced search operators. Simulated binary crossover exchanges the values between two parents based on the distribution index η_c, where a small value encourages more exploration (offspring are further from parents), while a large value encourages more exploitation (offspring are close to parents). Self-adaptation allows the evolutionary algorithm to dynamically adjust the distribution index (η_c) during the search process, based on the offspring performance.

Mutation operator. After the crossover operator, each chromosome undergoes a polynomial mutation operator [213] with p_{mut}. For each gene (variable) in the chromosome, a perturbation is introduced based on a polynomial distribution controlled by the mutation distribution index η_m. Larger values of η_m generate offspring closer to the original. This perturbation is applied with probability $\frac{1}{n_{var}}$, where n_{var} corresponds to the number of variables in the chromosome.

Duplicate removal. At each iteration, we remove the individuals that have a small deviation between them. We estimate the deviation as the cosine distance D_c [211] and compare it to a prefixed minimum threshold, D_{cth}. If the D_c between the two individuals is less than D_{cth}, one of them is removed.

The hyperparameters used for this GA configuration are specified in Table 5.3. We used the same hyperparameters for both test generation problems.

We present the overall process of the optimization in the latent space in Algorithm 2. Like every evolutionary algorithm, RILaST starts by creating an initial population P of size N_{pop}. Differently from existing initial population sampling techniques, in RILaST the population is created by randomly sampling L-dimensional vectors with values from a normal distribution. Then, each of the vectors in the population is assigned a fitness value based on the evaluation in the simulated environment. To perform the evaluation, each latent vector u is first decoded

Table 5.3 Hyperparameters of GA configuration for running RILaST

Parameter	Description	Values
N_{pop}	Population size	40
p_{cross}	Crossover rate	0.5
p_{mut}	Mutation probability	0.4
D_{cth}	Duplicate threshold	0.025
N_{off}	Number of offspring	20

with the decoder model M_D into the vectorized test scenario. We then use a transformation function ϕ_{tc} to map the test scenario from the vectorized form to the test scenario specification tc for the simulated environment. We run the simulation and compute the value of the objective function F_{sim}. In each of our case studies, we only have one objective function, but multiple objectives can be added if needed.

Algorithm 2 AmbieGenVAE: Test generation algorithm in the latent space

Require: Weights of the pre-trained VAE decoder M_D
Require: Simulator based evaluation function ϕ_{sim}
Require: Test scenario transformation function ϕ_{tc}
Require: Termination criterion T
Require: Hyperparameters $N_{\text{pop}}, N_{\text{off}}, p_{cross}, p_{mut}$
Set hyperparameters, initialize population P.
Load pre-trained VAE model
Initialize population P with N_{pop} individuals
Evaluate initial population:
for $n = 1$ to N_{pop} **do**
    Decode latent vector P_n into vectorized test scenario: $tc_{vec} \leftarrow M_D(P_n)$
    Convert vectorized test to original representation: $tc \leftarrow \phi_{tc}(tc_{vec})$
    Evaluate fitness: $F_{sim_n} \leftarrow \phi_{sim}(tc)$
end for
while not T **do**
    Select individuals for reproduction: $P_{\text{selected}} \leftarrow \text{TournamentSelection}(P)$
    Perform crossover: $P_{\text{crossover}} \leftarrow \text{Crossover}(P_{\text{selected}}, p_{\text{cross}})$
    Perform mutation: $P_{\text{offsring}} \leftarrow \text{Mutation}(P_{\text{crossover}}, p_{\text{mut}})$
    Evaluate offspring (as in initial population evaluation):
    for $n = 1$ to N_{off} **do**
        $tc_{vec} \leftarrow M_D(P_n)$
        $tc \leftarrow \phi_{tc}(tc_{vec})$
        $F_{sim_n} \leftarrow \phi_{sim}(tc)$
    end for
    Select survivors: $P \leftarrow \text{SurvivorSelection}(P, P_{\text{offspring}})$
end while
return final population P

When solutions are evaluated, tournament selection is used to choose the individuals for crossover and mutation, which are then performed on individuals with probabilities p_{cross} and p_{mut}, respectively. The obtained offspring $P_{\text{offspring}}$ are evaluated using the same process

as at the beginning of the search. We then use an elitist approach to select individuals for the next generation, where the N_{pop} individuals with the highest F_{sim} values are selected from the merged population of P and $P_{\text{offspring}}$. The search continues until a termination criterion is met. In our case, we used the number of evaluations or the running time as the termination criteria.

5.4 Evaluation

In this section, we formulate and answer the research questions related to the performance of RILaST. Our primary objective is to assess the extent to which the search with representation in the latent space of a VAE enhances the performance of test generators. To compare the algorithms, we assessed the following metrics: the average number of revealed failures (F_{av}) and their diversity. For the obstacle generation problem, we classify the test as failed if the SUT approaches the obstacles at a distance less than 1.5 meters. For the topology generation problem, a test is considered failed if more than 85 % of the SUT goes out of the lane bounds.

We evaluated the diversity of test scenarios based on the diversity of input test specifications. We did not consider the output diversity in terms of system behavior, as this metric is influenced by the stochasticity of the simulator. Intuitively, diverse input environment specifications should lead to diverse system behaviors. To evaluate the diversity, we leverage the failure sparseness, S, a metric proposed in the literature [81]. It estimates the diversity by computing the average value of the distance metric between each pair of the failed test cases tc:

$$S = \frac{\sum_i^N ave_j^N \text{dist}(tc_i, tc_j)}{N} \tag{5.5}$$

where N is the total number of failed test cases, *ave* represents the average value, and dist is the distance metric. In UC1, we used the cosine distance [211] as the distance metric *dist*. For UC2, we calculated *dist* as the weighted Levenshtein distance [180] between road segments near the location where the failure happened (car went out of the bounds) as suggested by Riccio and Tonella [81]. This metric was also used at the SBST 2021 tool competition [160].

5.4.1 Research Questions

We formulate the following research questions:

1. **RQ1** (Effectiveness of RILaST compared to baselines). *To what extent does RILaST improve the identification of diverse failures in autonomous robotic systems compared to baseline approaches?* In *RQ*1 we compare RILaST performance in terms of number

of failures found and their diversity to baseline approaches such as random search (RS) and genetic algorithm in original space as well as known approaches from the literature, such as AmbieGen [185], TUMB [214], RIGAA [215] and CRAG [186].

2. **RQ2** (Effectiveness of search in latent space). *To what extent does optimization in the latent space improve the identification of diverse failures compared to optimization with the original representation?* In this RQ, we compare the optimization in the original space with optimization in the latent space of two different VAEs: one trained with a randomly produced dataset and the other with an optimized dataset based on a simplified fitness function F_s.

3. **RQ3** (Choice of VAE hyperparameters). *What VAE configuration leads to the best performance in terms of reconstruction loss?* In this RQ we evaluate 18 VAE configurations with three different architectures, learning rate, and batch size values. Our goal is to identify the hyperparameters that lead to better performance. This aspect has not been thoroughly explored in previous work.

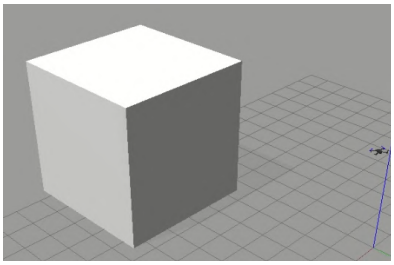
4. **RQ4** (Impact of the number of latent variables on the VAE performance). *What is the impact of different VAE configurations in terms of the latent space size on the resulting validation loss and reconstruction fidelity?* In this research question, we seek to explore how varying the number of latent dimensions of VAE during training impacts the model's performance. Specifically, we investigate how reducing as well as increasing the number of variables affects the final VAE loss as well as reconstruction fidelity.

5. **RQ5** (RILaST overhead). *What is the overhead of RILaST in terms of dataset collection time, VAE training and inference?* This research question focuses on evaluating the resource demands associated with training a VAE as part of the RILaST framework. We aim to quantify the overhead in three key areas: (1) the time required to collect and pre-process the dataset needed for VAE training, (2) the computational cost and time involved in training the VAE itself, and (3) the time taken for inference when generating scenarios in the latent space.

5.4.2 Systems Under Test

In our study, we consider two simulator-based autonomous robotic systems under test: an autonomous unmanned aerial vehicle in the Gazebo simulator [216] as well as an autonomous vehicle in the BeamNG simulator [148].

The autonomous UAV, shown in Figure 5.6a is an Iris 1 drone running PX4 2 control firmware



(a) UAV in Gazebo simulator

(b) AV in BeamNG simulator

Figure 5.6 Autonomous UAV and autonomous vehicle systems used in our study

[217]. This testing pipeline is based on the work of Khatiri et al. [118, 204]. The drone is tasked to navigate safely in a closed space with box-shaped obstacles. The drone is expected to follow a given trajectory autonomously, avoiding the obstacles. The goal of test generation is to reveal the scenarios where the UAV fails to complete the mission and approaches the obstacles at an unsafe distance of less than 1.5 m. This SUT corresponds to the first use case.

As the second test subject, we used an autonomous lane-keeping assist system agent BeamNG.AI, built-in in the BeamNG.tech [148] simulator, shown in Figure 5.6b. The goal of the agent is to navigate a given road without going out of its bounds. It has knowledge of the geometry of the entire road and uses a complex optimization process to plan trajectories that keep the ego-car as close as possible to the speed limit while staying inside the lane as much as possible. Test generation aims to find cases where the vehicle fails to follow the given trajectory. This SUT corresponds to the second use case we presented in Section 5.2.

5.4.3 Results

In this subsection, we present our experimental results for each RQ formulated above.

RQ1 – Effectiveness of RILaST compared to baselines

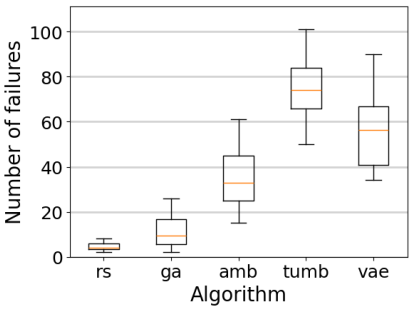
Motivation. In this research question, we aim to evaluate the ability of RILaST to generate diverse and challenging test scenarios. We compare RILaST to random search, GA in original space as well as to GA in original space guided by a combination of F_s and F_{sim}, where F_{sim} is used only for most promising tests according to F_s. This is an alternative way of embedding domain knowledge in the search process. We also compare RILaST with some of the known approaches from the cyber-physical system (CPS) testing literature.

Method. For both use cases, the baseline approaches include random search, a genetic algorithm, and the adapted AmbieGen method [218], where a simplified function is used to guide the search. In random search, test scenarios are generated by sampling randomly from the allowed parameter ranges specified in Table 8.1. The GA baseline follows the GA_L configuration for the search in latent space described in Section 5.3.1. However, instead of latent representation, it uses the same representation as GA for dataset collection GA_D. The next baseline, AmbieGen, uses both, simplified and simulator-based fitness functions to guide the search. In this method, F_{sim} is applied only when the F_s function predicts high fitness values, exceeding the established threshold. The simplified fitness function F_s was configured identically to the one used for RILaST dataset collection GA configuration GA_D. Further details about this baseline can be found in our report [218] for the SBFT Tool Competition 2024 CPS-UAV track [219]. As the further baselines, for UC1, we used the available approaches for UAV test generation, such as Tumb [214] tool, which showed one of the best results at the CPS-UAV testing tool competition in 2024 [219]. For UC2, we use the approaches for LKAS testing, namely RIGAA [215] and CRAG [79], which are the finalists of the SBFT CPS-ADS testing tool competition in 2024 [220].

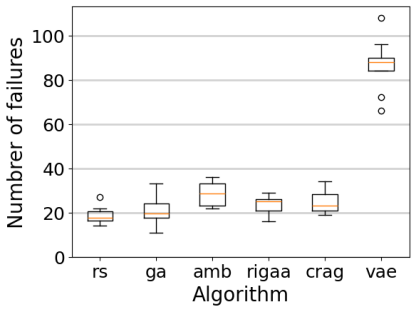
We compare the performance of RILaST to baseline methods in terms of the average number of revealed failures (F_{av}), as well as the average sparseness (S) of the failures. For the UC1, we allocate a budget of 200 evaluations to all the compared approaches, as it was done in the CPS-UAV testing competition. For the UC2, we allocate a two-hour time budget for each run of the algorithms. We run each baseline at least 10 times.

Results. The box plots for the average number of revealed failures and their sparseness for UC1 and UC2 are presented in Figure 5.7 and Figure 5.8, respectively. Tables 5.4– 5.5 report the obtained mean values and statistical test results. In the plots, we refer to AmbieGen as 'amb' and RILaST as 'vae'. For the rest of the tools, the name corresponds to their acronym.

For UC1, we can observe that RILaST outperforms both GA and RS in the number of revealed failures, identifying 4.64 times and 12.2 times more failures, respectively. While the sparseness of the test scenarios generated by RILaST is relatively high, it is, on average, 20% lower than that of randomly generated scenarios. RILaST reveals 59% more failures than the AmbieGen approach with no statistically significant difference in their diversity. Both approaches rely on the same simplified function to guide the test generation: RILaST during the dataset collection and AmbieGen during test execution. This result confirms that a smoother search space in the latent dimensions of the VAE positively influences the search process. TUMB reveals 33.4 % more failing test scenarios than RILaST. At the same

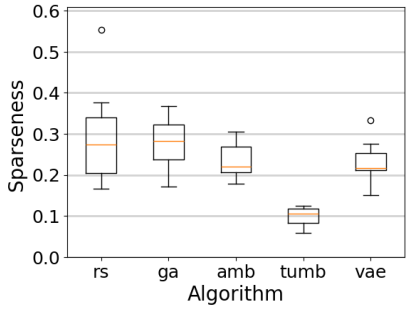


(a) Number of failures for UAV (UC1)

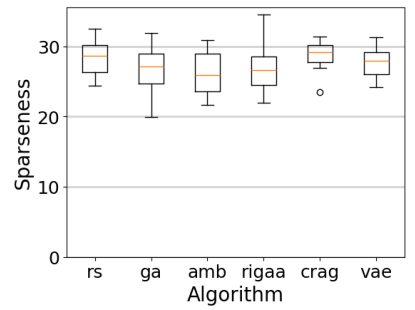


(b) Number of failures for ADS (UC2)

Figure 5.7 The average number of failures detected by different algorithms for the simulator-based autonomous robotic systems



(a) Sparseness of failures of the UAV



(b) Sparseness of failures of the ADS

Figure 5.8 The average sparseness of failures detected by different algorithms for the simulator-based autonomous robotic systems

Table 5.4 Number of revealed failures and their sparseness in uav case study

Metric	A	B	p-value	Effect size	Metric	A	B	p-value	Effect size
Failures	rs	ga	0.025	-0.6, L	Sparseness	rs	ga	0.97	-0.02, N
	rs	amb	<0.001	-1.0, L		rs	amb	0.307	0.273, S
	rs	tumb	0.001	-1.0, L		rs	tumb	<0.001	1.0, L
	rs	vae	<0.001	-1.0, L		rs	vae	0.254	0.286, S
	ga	amb	0.001	-0.855, L		ga	amb	0.084	0.455, M
	ga	tumb	0.001	-1.0, L		ga	tumb	<0.001	1.0, L
	ga	vae	<0.001	-1.0, L		ga	vae	0.05	0.486, L
	amb	tumb	0.001	-0.935, L		amb	tumb	<0.001	1.0, L
	amb	vae	0.007	-0.643, L		amb	vae	0.805	0.065, N
	tumb	vae	0.044	0.561, L		tumb	vae	<0.001	-1.0, L

time, the sparseness of the failing scenarios found by TUMB is 2.35 times lower than for the scenarios found by RILaST.

Table 5.5 Number of revealed failures and their sparseness in ads case study

Mertic	A	B	p-value	Effect size	Mertic	A	B	p-value	Effect size
Failures	rs	ga	0.343	-0.26, S	Sparseness	rs	ga	0.345	0.26, S
	rs	amb	0.001	-0.89, L		rs	amb	0.14	0.4, M
	rs	rigaa	0.03	-0.6, L		rs	rigaa	0.391	0.244, S
	rs	crag	0.004	-0.725, L		rs	crag	0.621	-0.133, N
	rs	vae	<0.001	-1.0, L		rs	vae	0.473	0.2, S
	ga	amb	0.011	-0.68, L		ga	amb	0.678	0.12, N
	ga	rigaa	0.219	-0.344, M		ga	rigaa	1	0.0, N
	ga	crag	0.074	-0.458, M		ga	crag	0.223	-0.317, S
	ga	vae	<0.001	-1.0, L		ga	vae	0.623	-0.14, N
	amb	rigaa	0.12	0.433, M		amb	rigaa	0.775	-0.089, N
	amb	crag	0.098	0.425, M		amb	crag	0.07	-0.467, M
	amb	vae	<0.001	-1.0, L		amb	vae	0.345	-0.26, S
	rigaa	crag	0.858	-0.056, N		rigaa	crag	0.166	-0.37, M
	rigaa	vae	<0.001	-1.0, L		rigaa	vae	0.488	-0.2, S
	crag	vae	<0.001	-1.0, L		crag	vae	0.199	0.333, M

For the ADS case study, RILaST generates, on average, from 3 to 4.7 times more failures than the rest of the tools. Specifically, it produces 3.68, 3.5, and 3 times more failures than the state-of-the-art tools RIGAA, CRAG, and AmbieGen, respectively. Meanwhile, all tools exhibit a high level of failure sparseness, with no statistically significant differences between them. On average, the CRAG tool achieves the highest sparseness value of 28.78.

Summary of RQ1: RILaST is effective in generating failure-revealing scenarios for autonomous robotic systems. In the UAV case study, it outperforms AmbieGen and GA by 59% and 464%, respectively. While RILaST generates 33% fewer failures than TUMB, its scenarios are 2.35 times more diverse. In the ADS case study, RILaST generates 3 to 4.7 times more failures than other tools, including 3.5 times more than RIGAA and CRAG, while maintaining a high level of failure sparseness.

RQ2 – Effectiveness of search in latent space

Motivation. This research question evaluates the effectiveness of test generation in latent space compared to the original space. A key advantage of the latent space is its regularization, which provides a smoother search landscape. Additionally, in the latent space, tests are sampled from a distribution similar to the training data, incorporating domain knowledge into the test generation process. This biases the search toward higher-performing solutions and reduces the test budget spent on non-promising cases. Here, we aim to understand the

impact of these benefits on test generation.

Method. To answer this RQ, we consider three different search spaces: the original space, the latent space of a VAE trained on randomly sampled tests, and the latent space of a VAE trained on tests previously optimized using a simplified fitness function. In each search space, we compare the performance of random search (denoted as 'ran' in the plots), genetic algorithm with problem-specific search operators (ga1), as well as the genetic algorithm with standard vector-based search operators (ga2). Random search samples the solutions uniformly from the allowed parameter ranges. Genetic algorithms $ga1$ and $ga2$ follow the configuration presented in Table 5.3 with different search operators. Algorithm $ga1$ uses problem-specific search operators as defined in Section 5.3.1. Algorithm $ga2$ uses simulated binary crossover as well as polynomial mutation [213], as defined in Section 5.3.3, which are state-of-the-art vector-based search operators. They can be used out-of-the box and do not need to be customized for a given problem. The original space *or* consists of solutions represented as vectors in N-dimensional space, as defined in Section 5.3.1.

The second search space is the latent space (vr) obtained by training a VAE on a dataset of 10,000 valid, randomly generated test scenarios. The corresponding search algorithms in this space are referred to as vr_ran, vr_ga1, and vr_ga2. Lastly, we consider the latent space of a VAE trained on test scenarios optimized with a simplified function F_s (vo), as described in Section 5.3.1. The search algorithms operating in this space are denoted as vo_ran, vo_ga1, and vo_ga2. For each configuration, we conduct 10 runs in the simulated environments corresponding to the two considered use cases, where the objective functions are evaluated based on the SUT's behavior in simulation. The test generation budget is the same as in RQ1, i.e., 200 evaluations and 2 hours for UC1 and UC2, respectively.

Results. Figures 5.9 and 5.11 illustrate the number of failures identified by different algorithms for the UAV and ADS case studies, respectively. Similarly, Figures 5.10 and 5.12 present the sparseness of the uncovered failures. Statistical test results are provided in Appendix B and summarized in Tables B.1 and B.2.

Firstly, we can observe that the search algorithms in the optimized latent space reveal more failures with a large effect size, compared to the search in original and random latent spaces for both use cases. Specifically, in UC1, the best-performing search algorithm in the vo space, vo_ga2, uncovers 3.6 and 3.48 times more failures than the best-performing algorithms in the or and vr spaces, respectively. In UC2, the vo space reveals 2.4 times more failures than vr and 4.14 times more than or.

At the same time, the algorithms in all the three spaces produce failures with high diversity, comparable to random generation (ran). In the UAV case study, only the vo_ga1 algorithm

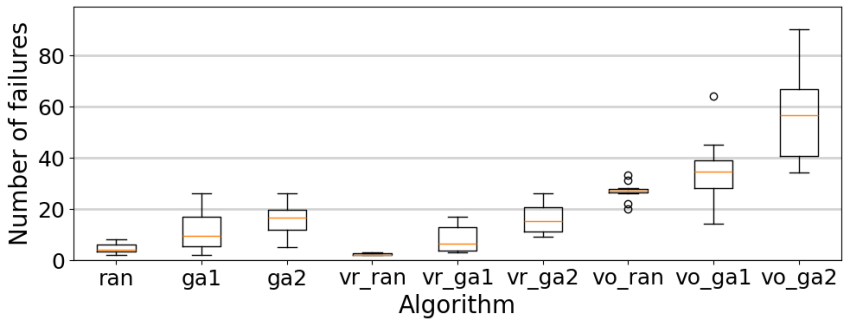


Figure 5.9 Number of failures for different search algorithms in different search spaces for UAV case study

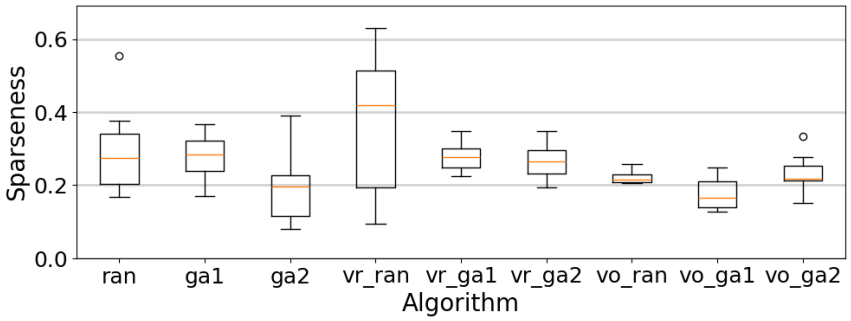


Figure 5.10 Sparseness of failures for different search algorithms in different search spaces for UAV case study

demonstrates a statistically significant reduction in sparseness, being 66% lower than *ran*. In the ADS case study, no statistically significant difference in sparseness is observed for algorithms in *vo* and *vr* compared to random search.

In the UAV case study, no clear advantage in terms of revealed failures is observed when comparing the original and random latent spaces. The algorithm *vr_ga*2 reveals 33% more failures than *ga*1, with no statistically significant difference in performance compared to *ga*2. However, in the ADS case study, *vr_ga*2 outperforms the best-performing algorithm in the original space, *ga*1, uncovering 71% more failures with a large effect size. Notably, *ga*1 relies on problem-specific search operators, which are time-consuming to design and fine-tune. In contrast, *vr_ga*2 leverages readily available, state-of-the-art search operators used in evolutionary search. This indicates that optimization in random latent space can be a promising approach, achieving effective search performance with standard search operators and without the need for developing problem-specific ones.

The effectiveness of state-of-the-art search operators, such as simulated binary crossover and

polynomial mutation, is further validated by the results in random and optimized latent search spaces. In both case studies, a GA employing these search operators outperforms a GA using domain-specific search operators. For the UAV testing, the algorithm vr_ga2 identifies 2.2 times more failures than vr_ga1, while vo_ga2 uncovers 1.56 times more failures than vo_ga1. Similarly, in the ADS use case, vr_ga2 identifies 1.44 times more failures than vr_ga1, and vo_ga2 uncovers 1.15 times more failures than vo_ga1. Interestingly, in the original space, the domain-specific search operators can be more effective. In the UAV case study, no statistically significant difference is observed between $ga1$ and $ga2$ performance. In the ADS case study, on the other hand, $ga1$ with domain-specific search operators outperforms $ga2$ with medium effect size.

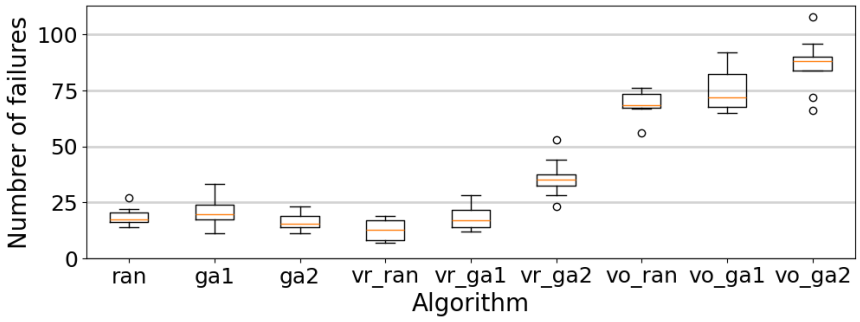


Figure 5.11 Number of failures for different search algorithms in different search spaces for ADS case study

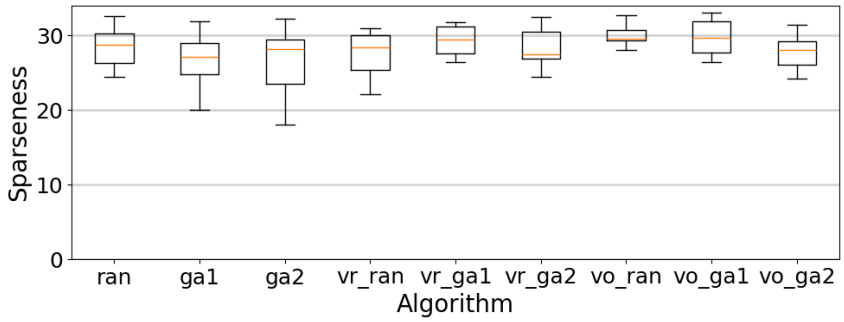


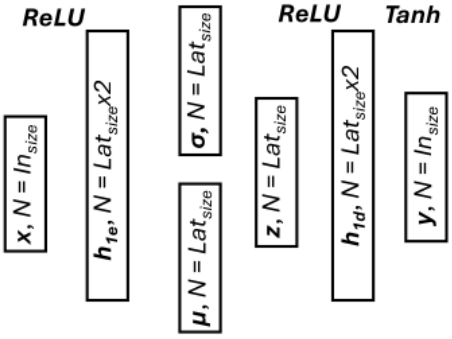
Figure 5.12 Sparseness of failures for different search algorithms in different search spaces for ADS case study

Summary of RQ2: For UC1, searching in the optimized latent space is 3.5 times more effective in revealing failures compared to the original and random latent spaces. For UC2, it is 2.4 times more effective than the random latent space and 4.14 times more effective than the original latent space. Algorithms in all three spaces generate failures with high diversity, comparable to random generation. In the ADS use case, optimization in the random latent space outperforms the original space by 71%, while no significant difference in the number of failures is observed for the UAV use case. GA with vector-based search operators uncovers more failures in the latent space with a large effect size compared to GA with problem-specific operators. However, in the original space, problem-specific search operators outperform vector-based ones with a medium effect size in the ADS use case.

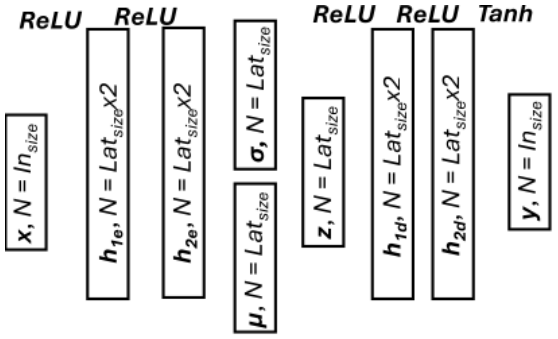
RQ3 – Choice of VAE hyperparameters

Motivation. Hyperparameter selection is an important step in the development of machine learning models. In this RQ, we aim to understand what VAE architecture and hyperparameters allow us to obtain a VAE with the lowest validation loss. Lower loss indicates that the VAE learned to perform the reconstruction with higher fidelity, which is important during the optimization process [127]. We focus our evaluation mainly on VAE architecture selection in terms of number of hidden layers and their size, which was not previously explored in the literature on optimization in latent space. We also experiment with batch size and learning rate, which are known to be sensitive hyperparameters [221].

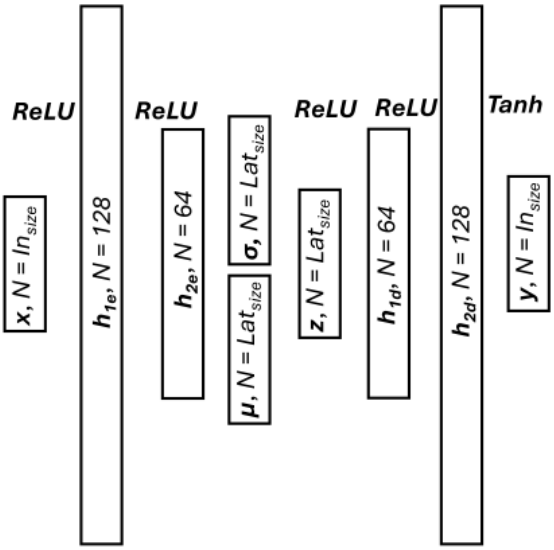
Method. In this RQ, we train VAEs using a dataset of optimized test scenarios. For training, we used Adam optimizer with first and second momentum coefficients set to 0.9 and 0.999, respectively, which are the recommended parameters in the literature [222]. We evaluated VAEs trained with different batch sizes (64, 128, and 512) and learning rates (0.001 and 0.0001). We train the VAE on the optimized dataset of 8000 training and 2000 validation entries. We considered three VAE architectures with different number of hidden layers and size sizes, as shown in Figure 5.13. The first VAE configuration, VAE1, is based on the setup proposed by Bentley et al. [127]. It includes one hidden layer and a latent space size that is double the input space size of 17, resulting in approximately 3019 parameters. The second configuration, VAE2, adds an additional hidden layer, nearly doubling the number of parameters to 5889. The third configuration, VAE3, consists of two hidden layers with sizes 128 and 64, respectively, and has a total of approximately four times the parameters of VAE2, i.e., 22435 parameters. Each hidden layer is followed by a Relu activation function.



(a) VAE1 architecture



(b) VAE2 architecture

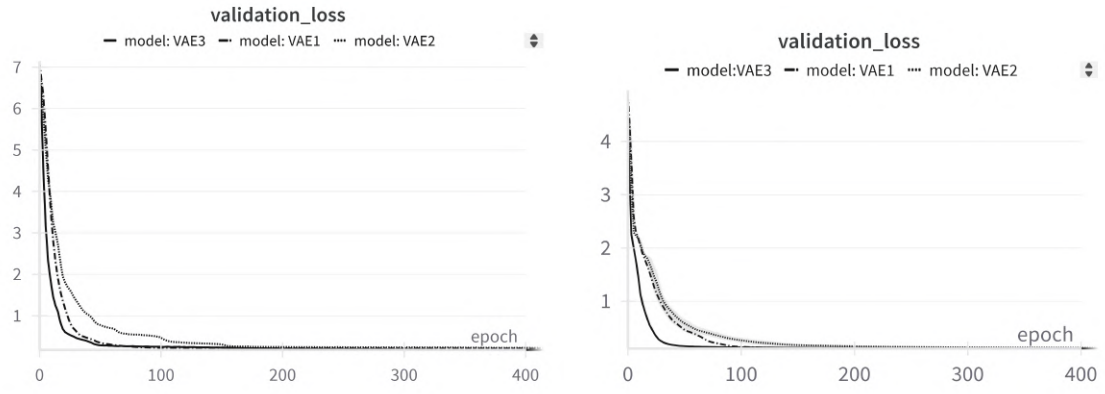


(c) VAE3 architecture

Figure 5.13 Different VAE configurations considered in hyperparameter fine-tuning

The inputs are normalized to the range between [-1, 1], and the $Tanh$ function is used at the output layer. In total, we evaluated 18 VAE configurations. For each configuration, VAE was trained three times for 1000 epochs. We report the average metrics, such as validation loss observed at the final epoch and training time. In all experiments, the size of the latent space z was fixed and equal to the input size.

Results. The obtained results are presented in Table 5.6. We observe that in both case studies, the configuration with $VAE3$, a learning rate of 0.001, and a batch size of 512 results in the lowest validation loss. Batch size has an important effect on the final loss and training time. For instance, in the UAV use case, with batch size decreasing from 512 to 128 and the rest of the parameters remaining fixed, the validation loss increases 2.79 times, and



(a) Convergence of VAE loss in for UAV (b) Convergence of VAE loss in for ADS

Figure 5.14 Convergence of VAE loss for different configurations with batch size of 512 and learning rate of 0.001

Table 5.6 Hyperparameter selection for VAE in UAV and ADS case studies.

UAV Use Case					ADS Use Case				
Model	**LR**	**Batch Size**	**Time (s)**	**Val. Loss**	**Model**	**LR**	**Batch Size**	**Time (s)**	**Val. Loss**
VAE3	0.001	512	1002.87	**0.139**	**VAE3**	0.001	512	932.17	**0.104**
VAE3	0.0001	512	1004.97	0.167	VAE1	0.001	512	883.38	0.109
VAE1	0.001	512	956.34	0.194	VAE2	0.001	512	907.16	0.111
VAE2	0.001	512	975.89	0.196	VAE3	0.0001	512	929.47	0.113
VAE1	0.0001	512	951.70	0.203	VAE1	0.0001	512	882.93	0.116
VAE2	0.0001	512	976.16	0.208	VAE2	0.0001	512	905.43	0.122
VAE3	0.001	128	1243.11	0.388	VAE3	0.001	128	1170.02	0.278
VAE3	0.0001	128	1245.90	0.429	VAE3	0.0001	128	1157.84	0.289
VAE1	0.001	128	1126.21	0.457	VAE2	0.001	128	1095.28	0.297
VAE2	0.001	128	1197.99	0.458	VAE1	0.001	128	1029.93	0.304
VAE1	0.0001	128	1132.14	0.471	VAE2	0.0001	128	1098.26	0.310
VAE2	0.0001	128	1192.60	0.488	VAE1	0.0001	128	1026.82	0.315
VAE3	0.001	64	1596.64	0.647	VAE3	0.001	64	1479.13	0.451
VAE3	0.0001	64	1553.38	0.681	VAE3	0.0001	64	1426.66	0.465
VAE2	0.001	64	1422.88	0.717	VAE2	0.001	64	1308.59	0.474
VAE1	0.001	64	1310.35	0.738	VAE1	0.001	64	1207.79	0.499
VAE1	0.0001	64	1310.20	0.744	VAE2	0.0001	64	1301.36	0.506
VAE2	0.0001	64	1428.39	0.748	VAE1	0.0001	64	1200.66	0.519

training time increases by 25%. Similarly, in the ADS use case, the validation loss increases 2.67 times and the training time increases by 25.5%. The increase in training time for lower batch sizes can be attributed to the additional computations required to calculate gradients and update the weights more frequently.

Examining the differences in VAE configurations for the UAV use case, the lowest loss achieved by VAE3 is 28.3% lower than that of VAE1 and 29% lower than that of VAE2.

Similarly, in the ADS use case, VAE3 achieves a 4.5% lower loss than VAE1 and 6.4% lower loss than VAE2. Furthermore, as shown in Figure 5.14, where all three VAE configurations use the same learning rate (0.001) and batch size (512), VAE3 demonstrates the fastest convergence.

Summary of RQ3: Overall, the best performance was observed with larger batch sizes (512) and learning rates of 0.001 for both case studies, with VAE configuration having bigger layer dimensions showing faster convergence and smaller final loss.

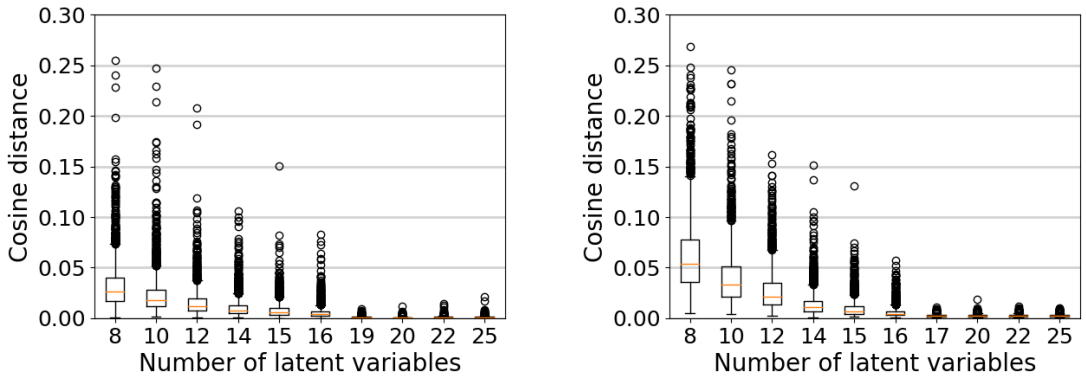
RQ4 – Impact of the number of latent variables on the VAE performance

Motivation. One of the key components of a VAE is the latent space. Maintaining the latent space dimension equal to the input size increases the likelihood of achieving a lower reconstruction error. However, reducing the latent space dimension decreases the problem's complexity, making optimization easier [128]. In this research question, we aim to explore the trade-off between latent space dimensionality and reconstruction fidelity.

Method. Having selected the key hyperparameters in RQ3, i.e., VAE3 configuration, 512 batch size, and 0.001 learning rate, we train VAEs with 10 different latent space sizes $nlat$, ranging from 8 to 25, for 1000 epochs. In each configuration, we train the VAE 3 times and select the VAE with the lowest validation loss (l_v). To verify that the VAE has learned a meaningful representation, we compare the distances between the initial test scenarios in the validation set (2,000 scenarios) and their reconstructed versions, i.e., the respective inputs and outputs of the VAE. Unlike validation loss, which considers both reconstruction error and KLD loss, distance metric gives a better evaluation of the fidelity of reconstruction. We use cosine distance c_d that we previously applied for duplicate removal and sparseness evaluation in the UAV case study. Experimentally, we established a threshold of 0.025 for identifying scenarios as duplicates. We thus expect the cosine distance between the input and reconstructed VAE test scenarios to be less than the established threshold. Otherwise, it signifies that the VAE has not learned a sufficiently meaningful representation and that reconstructed test cases differ from the original ones. We collect these metrics for two VAE types: one trained with a randomized dataset (VAE_R) and the other with the optimized dataset (VAE_O).

Results. The boxplots of cosine distance between original and reconstructed tests in validation set for VAE_R are shown in Figure 5.15. The values for the average cosine distance, validation loss, and training time are shown in Table 5.7. We can see that for both use cases,

the average cosine distance and validation loss decrease with bigger latent space size. At the same time, when the latent space dimension reaches the same size as the input, the validation loss and cosine distance values stabilize and don't show a noticeable reduction. The training time remains relatively stable, with a slight increase for latent space sizes, which are bigger than the initial ones. The boxplots further show that while the average cosine distance generally remains below the defined threshold, some reconstructions have cosine distances exceeding the threshold for latent dimensions of 16 or smaller. This suggests that VAE_R struggles to learn meaningful representations for certain tests when the latent space has a lower dimensionality.



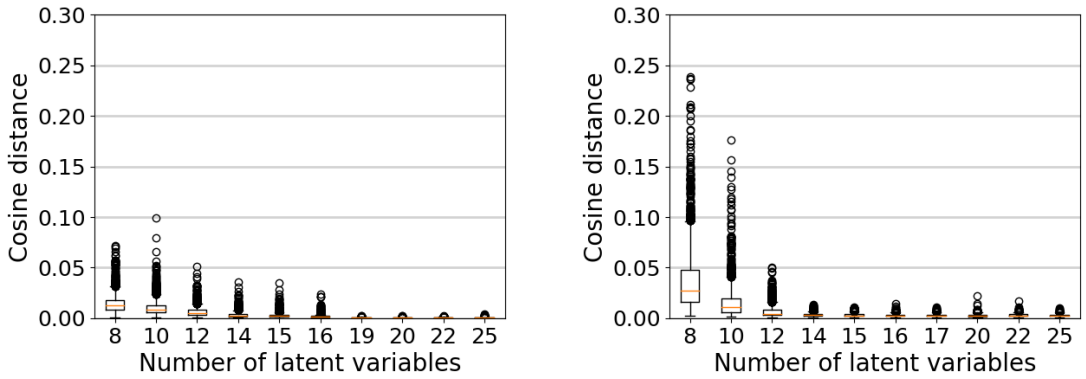
(a) Cosine distance for the UAV case study (b) Cosine distance for the ADS case study

Figure 5.15 Cosine distance between original tests and their reconstruction by the VAE based on a random dataset

Table 5.7 Evaluation of different latent space dimensions for random dataset based VAE

UAV use case					**ADS use case**			
nlat	**av dist**	**max dist**	**validation loss**	**training time**	**av dist**	**max dist**	**validation loss**	**training time**
8	0.03205	0.2551	1.7417	530.68	0.06236	0.333	0.6998	526.87
10	0.02362	0.2468	1.3662	530.02	0.04059	0.2454	0.5042	528.5
12	0.0159	0.2075	0.9356	528.51	0.02726	0.1619	0.3547	527.41
14	0.01077	0.1061	0.6649	531.06	0.01422	0.1515	0.2315	528.36
15	0.00838	0.1509	0.526	528.86	0.00967	0.1307	0.182	527.4
16	0.00568	0.0831	0.3782	**527.15**	0.00608	0.0568	0.1478	526.79
19	0.0009	**0.0088**	0.1604	527.35	**0.00249**	0.0113	**0.1112**	526.86
20	**0.00085**	0.0115	**0.1596**	532.94	0.00255	0.019	0.1116	**524.98**
22	0.00099	0.0141	0.1614	550.4	0.0026	0.0114	0.1117	536.55
25	0.001	0.0214	0.1667	539.43	0.00257	**0.0099**	0.1116	531.21

From Figure 5.16 and Table 5.8, which present the results for the VAE trained with the optimized dataset (VAE_O), we observe a similar trend to that of VAE_R. Specifically, the validation loss and cosine distance decrease as the number of latent variables increases, eventually plateauing when the number of latent variables ($nlat$) approaches the input size. How-



(a) Cosine distance for the UAV case study (b) Cosine distance for the ADS case study

Figure 5.16 Cosine distance for VAE based on optimized dataset with different sizes of the latent space

Table 5.8 Evaluation of different latent space dimensions for optimized dataset based VAE

UAV use case					**ADS use case**			
nlat	**av dist**	**max dist**	**validation loss**	**training time**	**av dist**	**max dist**	**validation loss**	**training time**
8	0.01443	0.072	1.219	532.048	0.0375	0.312	0.4686	529.469
10	0.01045	0.0994	0.866	531.720	0.0157	0.176	0.241	528.835
12	0.00642	0.0513	0.548	529.740	0.0065	0.05	0.1343	529.694
14	0.00347	0.036	0.363	531.321	0.0032	0.013	0.1103	530.276
15	0.00264	0.035	0.289	528.464	0.003	0.011	**0.1072**	528.498
16	0.00177	0.0241	0.22	**527.025**	0.0029	0.014	0.108	**519.634**
19	0.00059	0.0023	0.138	527.828	0.0029	0.011	0.1082	520.837
20	**0.00049**	0.0023	0.138	553.149	**0.0027**	0.022	0.1077	551.710
22	0.00061	**0.002**	0.141	563.320	0.0029	0.017	0.1079	534.286
25	0.00051	0.0044	**0.136**	549.533	0.0029	**0.01**	0.1086	525.334

ever, for the same value of $nlat$, the average validation loss and cosine distance are lower for VAE_O. For instance, given $nlat = 19$, in the UAV use case l_v is 14 % lower for VAE_O than VAE_R and d_c is 34.4 % lower. In the ADS use case, v_l is 2.7 % lower and no significant difference is observed for d_c. Bigger differences in l_v and d_c compared to respective values in VAE_R are observed from smaller values of $nlat$. In particular, for the UAV use case, the VAE with up to 16 latent dimensions (15.7 % reduction in size) does not exceed the c_d threshold. Similarly, in the ADS use case, the threshold is not exceeded for up to 14 (17.6 % reduction) latent dimensions. This indicates that a big portion of tests can be accurately represented with a smaller number of variables, simplifying the optimization process. Intuitively, the dataset for VAE_O contains less variety of inputs than VAE_R, making it easier to learn the representation. Finally, increasing the number of latent dimensions more than the original input size does not have a noticeable effect on reducing average v_l or c_d.

Summary of RQ4: Reducing the number of latent variables in a VAE increases both the validation loss and the cosine distance between the original and reconstructed tests. Conversely, increasing the number of latent variables beyond the input size does not significantly improve reconstruction accuracy. However, when an optimized dataset is used, the latent space size can be reduced by 15.7% for UAV testing and 17.6% for ADS without any noticeable reduction in reconstruction quality.

RQ5 – RILaST overhead

Motivation. In this RQ we evaluate the additional time overhead related to our RILaST approach. This includes the time to collect the dataset, train the VAE and perform the inference.

Method. For each use case, we record the time needed to generate random and optimized datasets, VAE training, and inference time. To measure the random dataset creation time, we generated 10 randomized datasets and recorded the average generation time. For the optimized dataset, to collect a dataset of 10,000 entries, we ran the genetic algorithm 50 times with a population of 200 for 50 generations, as described in Section 5.3. We recorded the average time it took to complete each run.

To evaluate VAE training time, we trained VAEs for each use case based on randomized and optimized datasets with latent space size corresponding to the input size. We ran the training 5 times for each configuration and recorded the average training time values. To record inference time, we used the trained VAE in each configuration during the optimization process and measured the time it takes to decode a test. We recorded the values for 100 inferences. In the results section, we report the average value of time taken as well as the standard deviation (in brackets).

Table 5.9 Overhead of training VAE with random generated dataset

UAV use case			ADS use case		
Dataset collection, s	VAE training, s	VAE inference, s	Dataset collection, s	VAE training, s	VAE inference, s
41.692 (1.85)	527.34 (3.17)	0.0011 (0.00021)	90.32 (2.13)	526.86 (2.796)	0.00107 (0.00023)

Table 5.10 Overhead of training VAE with optimized dataset

UAV use case			ADS use case		
Dataset collection, x0.02 s	VAE training, s	VAE inference , s	Dataset collection, x0.02 s	VAE training, s	VAE inference , s
338.52 (53.173)	527.828 (2.16)	0.0011(0.00026)	66.732 (7.008)	520.83 (10.27)	0.00115 (0.00038)

Results. From Table 5.9 we can see that the biggest time overhead when training the VAE with randomized dataset is in the dataset collection phase, taking on average 527 seconds in both use cases. The data collection time is around 42 seconds for the UAV use case, taking more than twice as long (90 seconds) in the ADS use case. Data inference time takes on average 0.001 s per evaluation in both use cases. Considering that typically no more than 1000 evaluations will be used in a single run, this time remains negligible.

For VAE with optimized dataset, the biggest overhead is in the dataset collection phase, as it can be seen from Table 5.10. Genetic algorithm with 50 generations and a population size of 200 takes an average of 338.5 seconds per run in the UAV use case. Given a dataset size of 10,000, fifty runs are required, resulting in a total collection time of approximately 4.7 hours. In contrast, data collection for the ADS use case is five times faster, with each run taking an average of 66 seconds and a total dataset collection time of about 0.9 hours. We attribute this to the fact that the simplified objective function for ADS is less computationally expensive. Although the dataset collection time is substantial, it can be significantly reduced, as genetic algorithm runs are highly parallelizable. Currently, no parallelization techniques have been applied. The VAE training times are comparable to those observed with the random dataset-based VAE, averaging approximately 527 seconds. VAE inference remains minimal, requiring only 0.001 seconds per evaluation.

Summary of RQ5: VAE training introduces the largest time overhead, with the random dataset-based VAE requiring 527 seconds on average. Dataset collection times are relatively low, ranging from 40 to 90 seconds. Inference time is negligible, taking less than 10 milliseconds. However, for the optimized dataset-based VAE, dataset collection time can be substantial, depending on the computational cost of the simplified objective function. In the UAV use case, collection took 4.7 hours, whereas in the ADS case study, it required only 0.9 hours. This time can be significantly reduced, as the genetic algorithms used for dataset collection are highly parallelizable.

5.4.4 Threats to Validity

Internal validity. To minimize the threats to internal validity, relating to experimental errors and biases, whenever available, we used standardized frameworks for development and evaluation. We implemented the evolutionary search algorithms based on a popular Python-based Pymoo framework [179]. We trained the VAE based on a commonly used machine learning framework Pytorch [223]. For the autonomous drone use case, we used Aerialist benchmark proposed by Khatiri et al. [84], which was also utilized at the SBFT

2024 UAV tool competition [85]. To evaluate the scenarios for the LKAS case study, we leveraged a standardized test pipeline from the SBST 2022 workshop tool competition [155]. To minimize the threats to internal validity stemming from selected parameter values, we conducted extensive experimentation with various hyperparameters and configurations for VAE training.

Conclusion validity. Conclusion validity is related to random variations and inappropriate use of statistics. To mitigate it, we followed the guidelines in [192] for search-based algorithm evaluation. We repeat all evaluations a sufficient number of times to ensure statistical significance. For all results, we report the non-parametric Mann-Whitney U test with a significance level of $\alpha = 0.05$, as well as the effect size measure in terms of Cliff's delta.

Construct validity. Construct validity is related to the degree to which an evaluation measures what it claims. In our experiments, we used two established evaluation metrics: number of revealed failures and their sparseness. For ADS, we used the failure and sparseness definition established in SBST 2022 CPS testing tool competition. For the autonomous UAV, we used the same failure metric definition as in SBFT 2024 UAV test generation competition. For the evaluation of diversity, we utilized cosine distance, which is a known metric for comparing the difference between vectors [224]. Prior to using the metric, we ensured that the vectors representing failed test scenarios are properly normalized. Finally, we conclude that our proposed RILaST approach outperforms the available baselines in terms of the number of revealed failures. We acknowledge that this finding is true, provided that RILaST is allocated a one-time budget to generate the dataset and train the VAE.

External validity. External validity relates to the generalizability of our results. We demonstrated how our framework can be applied to the generation of test environments for two different autonomous robotic systems and two test generation tasks. However, we only considered a limited number of test subjects and limited levels of environment complexity. To further explore the generalizability of RILaST, additional experiments with a broader range of robotic systems and more complex environments would be beneficial.

5.4.5 Data Availability

The replication package of our experiments, including the implementation of RILaST and the baseline algorithms is available at Zenodo repository [201] as well as GitHub: `https://github.com/swat-lab-optimization/RILaST`.

5.5 Discussion

This section presents the key lessons learned and main recommendations for using RILaST. It also outlines the limitations of RILaST and the challenges encountered when training VAEs for test generation.

5.5.1 Usefulness of RILaST and Main Insights

As seen from the Results section, performing the search in latent space is beneficial for test generation. The simplest way to apply RILaST is by training a VAE using a randomly generated dataset, which incurs a low overhead of approximately 10 minutes on average. By leveraging the trained VAE, optimization can be performed in the latent space, eliminating the need to design problem-specific search operators. Instead, readily available vector-based search operators can be used directly. RILaST achieves comparable performance to genetic algorithms (GA) with custom search operators, and can outperform them by up to 71% in terms of the number of failures revealed.

When a simplified fitness function is available, it can guide dataset collection for VAE training, biasing the latent space toward higher-performing solutions and further facilitating the search. This approach increases the number of revealed failures from 4.14 to 4.68 times compared to the original space without compromising diversity. While training a VAE on an optimized dataset may introduce a noticeable overhead, it is a one-time cost that can be significantly reduced using parallelization techniques, such as running the GA in different computational threads for dataset collection.

When an optimized dataset is used, the dimensionality of the search space can be reduced from 15.7 % to 17.6 % facilitating the search process without introducing noticeable reconstruction errors. When training a VAE, architectures with bigger hidden layers converge faster to lower loss values, although no significant difference is observed in the final loss. Learning rate of 0.001 and batch size of 512 show the best performance.

5.5.2 Interpreting VAE Latent Dimensions

The purpose of training the VAE is to learn meaningful representations of tests in the latent space. To visually analyze the learned representation, we sampled random tests in the latent space and fixed the values of all dimensions except one, which we varied within the range of -3σ - 3σ. Figure 5.17 demonstrates an example of the learned representation of a test in the UAV case study. By varying the value of the 8th dimension from -2.38 to 2.38, we observe changes in an obstacle's rotation and width. This suggests that modifying a single latent

variable can correspond to transformations involving multiple variables in a latent space.

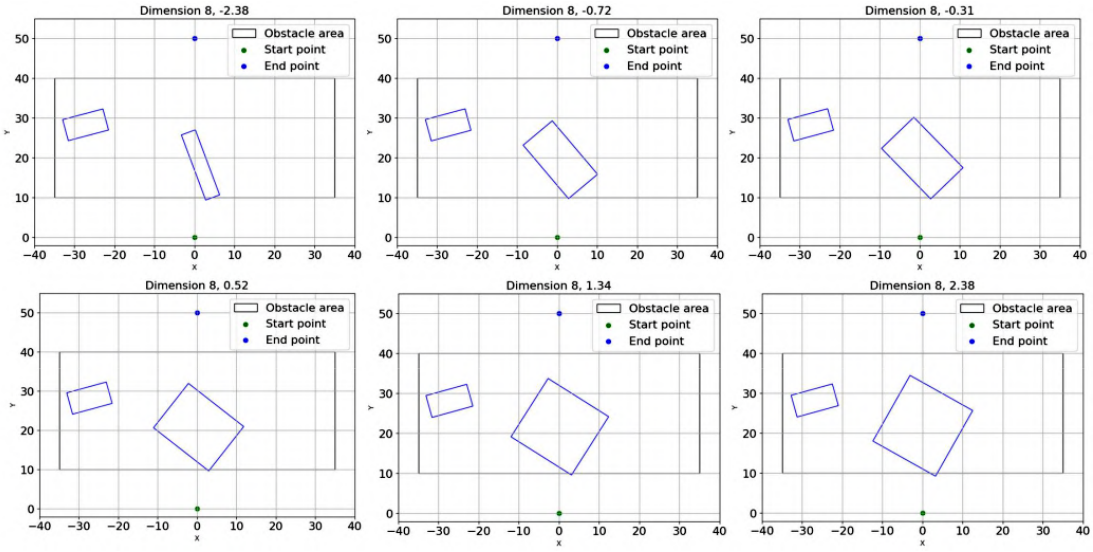


Figure 5.17 Representation in a latent space for different values of dimension 8 in the UAV use case

Figure 5.18 shows an example of the learned representation in UC2, when the dimension 3 is varied. We can observe a complex transformation, where the 'left' turn transitions to a 'right' turn. Such transformation would involve some complex manipulation of a number of values in the original space (curvature values). Based on these examples, we argue that the latent space representation can facilitate non-linear transformations of test scenarios that would be difficult to achieve in the original space. This approach could potentially enable more complex transformations, such as modifying non-rectangular obstacle shapes. Additionally, it may be useful for transforming test scenarios with complex representations, such as 3D meshes.

5.5.3 Limitations of RILaST

In this subsection, we discuss key challenges related to test generation in the latent space. One of the main challenges is designing the loss function for training the VAE. The loss consists of two components: one accounting for reconstruction error and another enforcing Kullback–Leibler divergence to ensure that the latent space follows a normal distribution. However, in test generation for autonomous robotic systems, it is crucial to ensure the validity of the generated tests. In other words, the generated tests must satisfy the pre-defined constraints C. Current VAE training techniques do not directly account for invalid tests and may result in a latent space containing invalid tests. Figures 5.19 and 5.20 show examples

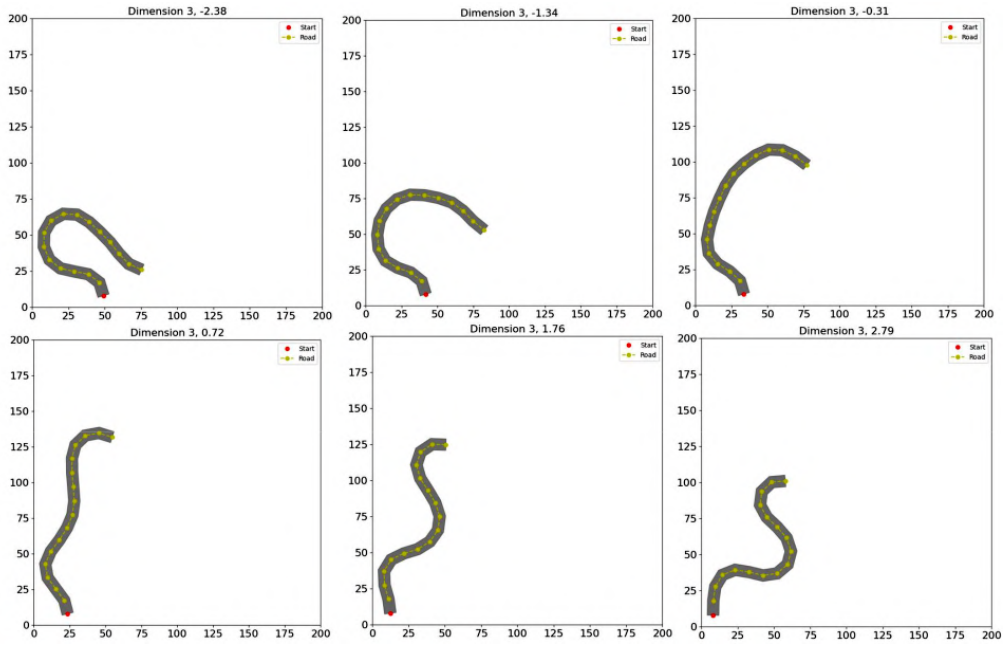
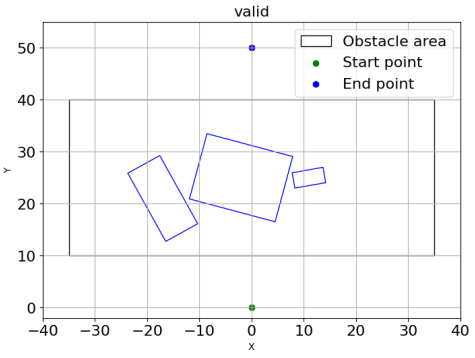
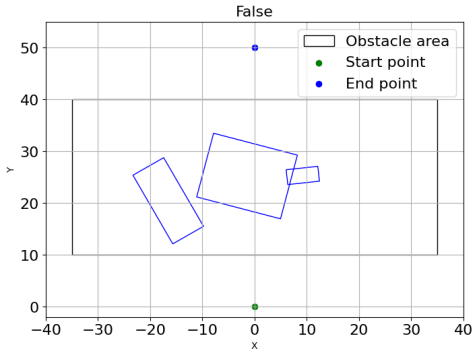


Figure 5.18 Representation in a latent space for different values of dimension 3 in the ADS use case

of original (left) and VAE-reconstructed (right) tests that show high similarity and consequently a low reconstruction loss. However, while the original tests on the left are valid, the reconstructed tests on the right are invalid, as they do not satisfy the constraints. In Figure 5.19b we can see intersecting obstacles and in Figure 5.20b the produced road topology is too sharp. We thus argue that there is a need to modify the loss functions by adding



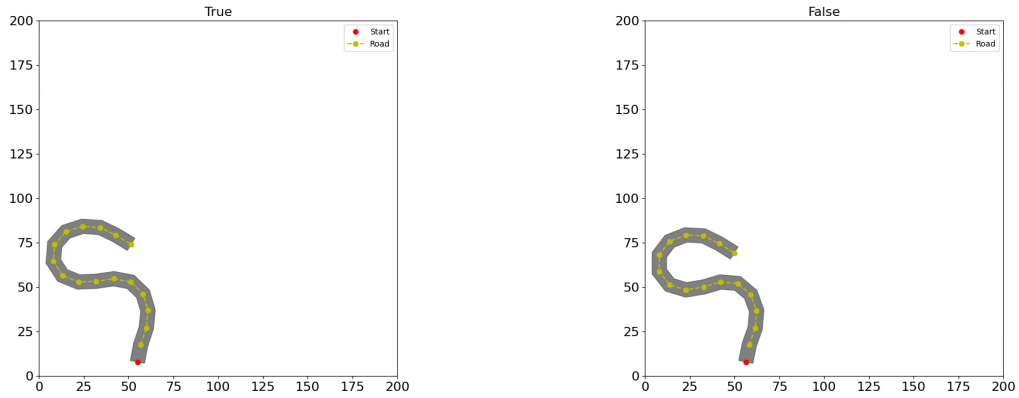
(a) Original test scenario for UAV (valid)



(b) Reconstructed test scenario for UAV (invalid)

Figure 5.19 Example of a valid original scenario and invalid reconstructed for the UAV use case

regularization terms specific to constraint satisfaction in order to encourage the VAE to learn only valid tests. Invalid tests produced in a latent space are not critical, as oftentimes there is an inexpensive function available that checks for test validity prior to its execution in the simulator. We surmise that having a latent space of only valid test scenarios could further improve the search and test generation in a latent space.



(a) Original test scenario for ADS (valid) (b) Reconstructed test scenario for ADS (invalid)

Figure 5.20 Example of a valid original scenario and invalid reconstructed for the ADS use case

5.6 Chapter Summary

In this chapter, we presented a novel approach, **RILaST**, for test representation enhancement in a latent space. Based on experiments with two systems under test, a UAV obstacle avoidance system and a lane keeping assist system, we have shown that search in an optimized latent space leads to the discovery of more failures compared to search in the original space. Moreover, representing tests in a latent space enables the learning of non-linear test transformations, which would otherwise require complex processing in the original space. We explored two types of latent spaces: one obtained from a dataset of randomized test scenarios and another from a dataset of optimized tests selected based on a known heuristic, i.e. operational knowledge. The first latent space allows for the discovery of up to 71% more failures compared to the original space, while also eliminating the need for custom search operators. The optimized latent space further improves the search process, leading to the discovery of 3.5 to 4 times more diverse failures compared to the original space. Interestingly, RILaST also outperformed the baseline, where both simplified (same as for RILaST search space improvement) and full objective functions were used for evaluation, revealing from 1.8 to 2.6 times more failures.

These findings indicate that optimization in a latent space is an effective technique for improving search-based testing, particularly when a simple heuristic for identifying challenging test scenarios is available. The one-time computational overhead of RILaST can be reduced by leveraging parallelization techniques.

CHAPTER 6 DYNASTO: VALIDITY-AWARE DYNAMIC–STATIC PARAMETER OPTIMIZATION FOR AUTONOMOUS DRIVING TESTING

6.1 Problem Context

Autonomous driving systems (ADS) must operate safely in highly dynamic traffic environments, where rare yet safety-critical situations emerge from complex interactions between vehicles [21]. Simulation-based testing is widely used to assess ADS behavior [22], motivating a growing body of work on automatic scenario generation, where adversarial agents or search procedures construct traffic situations that expose weaknesses in the system under test. Early methods primarily searched over static initial conditions, such as the starting positions and speeds of traffic participants [39, 225], or optimized pre-defined trajectories [195, 226]. A key limitation is that the adversary's behavior is fixed a priori and remains non-reactive to the SUT during execution, which reduces its ability to provoke realistic, interactive failures.

More recent approaches use reinforcement learning to learn adversarial driving policies or disturbance sequences that adapt online and steer the SUT toward failures [30, 36, 134] by rewarding collision provoking behaviors. While this dynamic adaptation can substantially improve failure discovery, RL-based testing raises two persistent challenges addressed by DYNASTO. First, it is difficult to design rewards that discourage reward-hacking behavior: adversaries may trigger collisions via unrealistic, overly aggressive maneuvers, and many existing testing methods treat all collisions as equally valuable [30, 36]. This can obscure whether a collision reflects a genuine SUT brittleness, for example, failing to avoid a crash when a lead vehicle brakes from a safe initial headway, or an invalid interaction, for example, a cut-in at an unsafe distance that leaves the SUT insufficient time to react. Second, RL-based methods often operate under a fixed or narrowly sampled distribution of static scenario parameters (e.g., initial lanes, gaps, and speeds). Prior work in search-based testing [39, 95] has shown that varying these initial conditions can be decisive for uncovering additional safety-critical failures.

In this work, we propose DYNASTO (Validity-Aware Dynamic-Static Parameter Optimization for Autonomous Driving Testing), a two-step approach that jointly searches over dynamic adversarial behaviors and static scenario parameters and optimizes them in a coordinated way. In Step 1, we train an adversarial agent using RL to generate dynamic behaviors that challenge the ego vehicle (the SUT). The reward function is defined using temporal-logic-based validity criteria and a safe-distance model inspired by established safety standards (e.g., ISO 34502), so that only failures induced by unsafe ego behavior and respecting a

minimum safe distance are promoted. Invalid failures due to unrealistically close or physically implausible maneuvers by the adversary are discarded. In Step 2, we run an evolutionary search over the initial conditions (e.g., initial positions, lanes, and speeds) of the failing scenarios identified in Step 1. Replaying the same failure-inducing behaviors under different static configurations allows revealing additional failures that were not found during the initial adversarial agent training phase.

To support post-hoc analysis, DYNASTO includes a graph-based clustering procedure that groups failures into higher-level modes. Each failure trace is mapped to a sequence of discrete semantic events (e.g., cut-ins, cut-outs, and braking interactions), which is used to construct a k-nearest neighbors graph. We then apply the Leiden community detection algorithm [227] to identify clusters of failures that share similar interaction patterns, reducing hundreds of individual failures to a smaller number of representative patterns. We manually analyze the failures in the discovered clusters and highlight systematic weaknesses in the ego controller.

We evaluate DYNASTO in the *HighwayEnv* simulator [228] on two systems under test (SUT1 and SUT2), corresponding to ego vehicles trained in two different environments. *HighwayEnv* is a 2D simulator commonly used in the RL community for self-driving policy training [**?**]. We compare our approach against random adversarial actions, a genetic algorithm (GA) that searches over fixed sequences of adversarial actions, the Deep Q-Learning [49] baseline of Doreste et al. [30] that promotes all failures equally, and our own Deep Q-Network (DQN) adversary with a validity-aware reward. We further investigate how to combine dynamic action optimization with RL and search over static initial scenario parameters with GA. Specifically, we compare (i) co-evolutionary designs in which an RL adversary and a GA jointly optimize actions and initial conditions in parallel, and (ii) sequential designs in which the GA explores initial conditions only after the RL adversary has been trained and its failure-inducing behaviors can be replayed. Across these settings, DYNASTO finds approximately 60%–70% more valid failures than a DQN-only adversary under the same evaluation budget and yields about 20% more failure clusters than the co-evolutionary baseline. Finally, our post-hoc clustering analysis compresses roughly 200 unique failures per SUT into about 12 representative patterns.

In summary, this chapter makes the following contributions:

- We introduce a validity semantics that separates genuine ADS faults from unrealistic crashes, via automatic trace labeling using Signal Temporal Logic (STL) rules aligned with ISO 34502 two-vehicle highway scenarios.
- We present DYNASTO, a two-step scenario-generation approach that couples RL-based adversarial action optimization with evolutionary search over initial conditions, guided

by temporal-logic validity constraints.

- We propose a graph-based clustering analysis that groups failures into interpretable modes using semantic event sequences, enabling scalable and structured failure analysis.
- We evaluate two autonomous driving controllers in *HighwayEnv*, comparing DYNASTO against Random Search (RS), GA-only, and RL-based baselines, as well as co-evolutionary RL+Search Algorithm variants: GA or RS.
- We analyze representative failure modes uncovered by DYNASTO, showing how it exposes systematic weaknesses in ego-vehicle behavior.

6.2 Approach

This section presents DYNASTO, a validity-aware traffic scenario generation workflow that jointly optimizes *dynamic adversarial behaviors* and *static initial conditions* (Figure 6.1). DYNASTO formalizes *valid* failures with temporal-logic rules that filter out collisions caused by overly aggressive or unrealistic adversarial behavior, retaining only safety-relevant failures consistent with safe-distance constraints and scenario-specific interaction patterns. Step 1 trains an RL adversary to generate disturbance strategies that challenge the ego vehicle (SUT) while being rewarded only for valid failures, producing a set of failure traces with the adversary's action sequences. Step 2 then searches over static initial parameters with a GA, evaluating each candidate by replaying the Step 1 adversary actions while the SUT responds online and re-checking validity under the same criteria. Finally, DYNASTO enables post-hoc failure de-duplication and clustering to summarize failures into representative modes for analysis.

6.2.1 Prerequisite: Valid Failure Modeling

DYNASTO relies on a precise failure definition. We target collisions that arise from behaviorally plausible, safety-relevant interactions, rather than from trivially unsafe or physically implausible trajectories. Central to this distinction is a *safe-distance* requirement: the adversary must initiate its maneuvers with sufficient longitudinal headway so that, under normal conditions, the ego vehicle has enough time to react and avoid a collision. Collisions that occur despite this requirement are labeled *valid* failures, while collisions triggered by violating the safe-distance condition are labeled *invalid* and excluded from downstream analyses.

To bound the space of critical interactions, we follow ISO 34502 [229], which standardizes highway test scenarios for autonomous vehicles. The standard defines 24 critical scenarios,

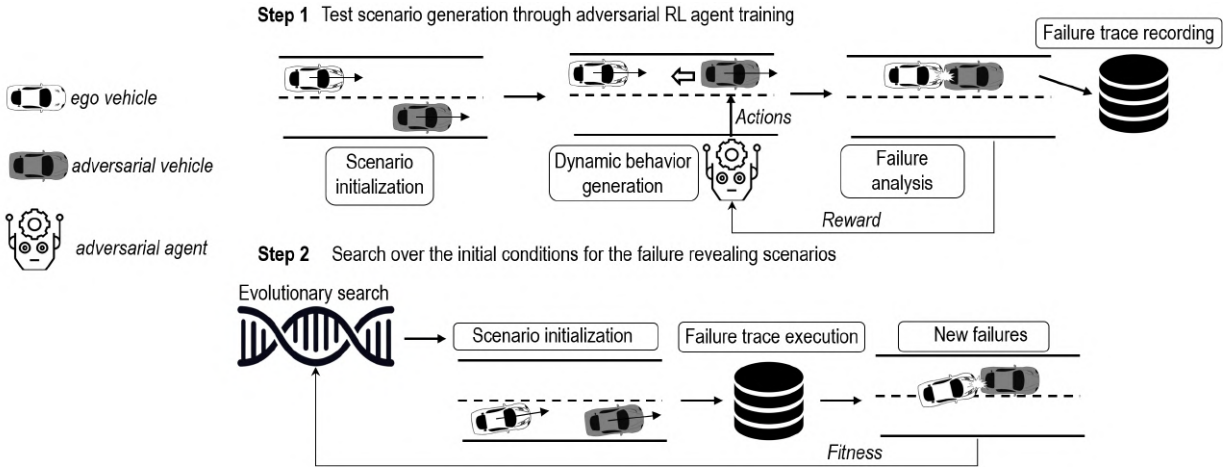


Figure 6.1 The DYNASTO Two-Step Approach Diagram

including 21 two-vehicle cases; we therefore prioritize two-vehicle scenarios. These scenarios span several zones, including the main highway, merge, and departure zones. In this work, we focus on two-vehicle scenarios in the main highway zone, leaving the remaining zones for future work. Our objective is to ensure that adversarial disturbances provide the SUT sufficient time to react and are not excessively aggressive. To this end, we focus on four ISO 34502 scenarios in which the adversary, also referred to as the principal other vehicle (POV), initiates the maneuver.

When a collision occurs, the trace is analyzed to determine whether the POV satisfied the corresponding safety constraints prior to the collision. If the constraints are respected, the failure is classified as valid; otherwise, it is considered invalid and attributed to overly aggressive POV behavior. We automate this trace classification using Signal Temporal Logic (STL), a common formalism for reasoning about cyber-physical system signals [78]. STL formulae are defined over predicates on signal variables and combine them using Boolean and temporal operators, including *eventually* ($\Diamond_{\mathcal{I}}$), *always* ($\Box_{\mathcal{I}}$), and *until* ($\mathcal{U}_{\mathcal{I}}$), where the interval $\mathcal{I}$ specifies timing constraints.

The ISO 34502 scenarios selected for detailed analysis are shown in Figure 6.2. Scenarios *a*–*b* correspond to collisions that occur after the POV changes lanes. In scenario *a*, a failure is considered valid only if the safe longitudinal distance D_{lon} is respected at the moment of the cut-in maneuver. In contrast, scenario *b* is always labeled invalid, since the required safe lateral distance D_{lat} is violated during the cut-in. The definitions and computation of D_{lon} and D_{lat} are discussed in Section 6.2.2. These distances specify the minimum admissible longitudinal and lateral separation required to ensure the following vehicle has sufficient time to react to the POV's maneuver.

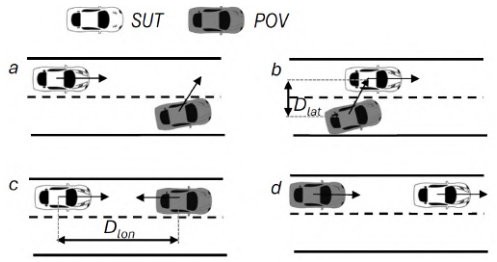


Figure 6.2 Critical scenarios with maneuvers initialized by POV

To process the failure traces with STL rules, we first define a set of low-level predicates over the relative positions and accelerations of the ego (SUT) and adversary (POV) vehicles:

$$
\begin{aligned}
\mu_{\text{unsafe}} &:= \big(d_{\text{lon}}(t_m) < d^{\text{lon}}_{\text{safe}}(t_m) \wedge \mathit{SameL}\big) \vee \\
&\qquad \big(d_{\text{lat}}(t_m) < d^{\text{lat}}_{\text{safe}}(t_m) \wedge \neg \mathit{SameL}\big); \\
\mathit{Ahead}_{adv} &:= x_{\text{adv}}(t) > x_{\text{ego}}(t); \\
\mathit{LaneC}_{adv} &:= (\text{prev}(\text{lane}_{adv}) \neq \text{lane}_{adv}) \wedge (\text{prev}(\text{lane}_{ego}) = \text{lane}_{ego}); \\
\mathit{SameL} &:= |y_{\text{adv}} - y_{\text{ego}}| < \delta_{\text{lat}}, \text{ same lane}; \\
\mathit{Brake}_{adv} &:= a_{\text{adv}} \leq -a_{\min}; \\
\mathit{Accel}_{adv} &:= a_{\text{adv}} \geq a_{\min}; \\
\mathit{CutIn}_{adv} &:= \Diamond_{[t,t+\Delta t]}\Big(\mathit{LaneC}_{adv} \wedge \mathit{Ahead}_{adv} \wedge \mathit{SameL}\Big).
\end{aligned}
$$

Here t_m corresponds to a timestep where one of the unsafe maneuvers shown in Figure 6.2 $(a-c)$ was initiated, δ_{lat} a small threshold to determine if the two vehicles are in the same lane. Firstly, the STL evaluation is applied only to traces that result in a collision, i.e., failures involving the two vehicles. Accordingly, each trace is analyzed over the interval $I \in [0, T_c]$, where T_c refers to a timestep at which the collision occurs.

In the specified predicates, μ_{unsafe} detects whether the safe lateral or longitudinal distances are preserved, *CutIn* determines if the cut-in maneuver was performed, *Ahead* and *Behind* whether the POV vehicle is ahead or behind the SUT, *SameLane* detects if POV and SUT are in the same lane, *Brake* and *Accel* detect whether the POV is breaking or accelerating.

Based on the defined predicates, we formulate the following STL rules for the trace analysis:

$$\varphi_{\text{unsafe-cut-in}} := \Diamond_{[0,\,T_c]}\big(\mathit{CutIn}_{adv} \wedge \mu_{\text{unsafe}}\big) \tag{6.1}$$

$$\varphi_{\text{unsafe-brake}} := \Diamond_{[0,\,T_c]}\big(\mathit{Ahead}_{adv} \wedge \mu_{\text{unsafe}} \wedge \mathit{Brake}_{adv} \wedge \mathit{SameL}\big) \tag{6.2}$$

$$\varphi_{\text{rear-hit}} := \Diamond_{[0,\,T_c]}\Big(Ahead_{ego} \wedge Accel_{adv} \wedge SameL\Big) \tag{6.3}$$

These STL specifications are evaluated on simulated traces to automatically detect and classify interactions. We use the `RTAMT` Python library [230] to specify and evaluate the rules. If any rule evaluates to true, the collision is labeled *invalid* and discarded; otherwise, it is labeled a *valid* failure.

This labeling mechanism is important for the rest of DYNASTO. In Step 1, it determines which outcomes contribute to the validity-aware reward used for adversarial training, and in Step 2, it provides the fitness signal for searching over initial conditions. As a result, both steps focus on behaviorally meaningful, safety-relevant failures rather than artifacts of unrealistic adversarial behavior.

6.2.2 Step 1: Adversarial Policy Learning

In Step 1, DYNASTO learns adversarial driving behaviors that trigger *valid* failures as defined in Section 6.2.1. We cast the adversary as a reinforcement learning (RL) agent, encoding failures and validity into a scalar reward and updating the policy to maximize their occurrence. This formulation allows the adversary to generate temporally extended interaction patterns, such as sequences of lane changes and speed adjustments, conditioned on the current traffic context.

Action space

The adversarial agent operates in the discrete meta-action space provided by the `highway` environment in the `HighwayEnv` simulator. In this setting, the SUT drives on a two-lane highway with one adversarial vehicle. The SUT aims to maintain high speed while avoiding collisions. Actions are specified via the `DiscreteMetaAction` interface, which maps a small set of high-level driving intentions to predefined low-level vehicle controls. The action space contains five discrete actions: `LANE_LEFT`, `IDLE`, `LANE_RIGHT`, `FASTER`, `SLOWER`. These allow the adversary to change lanes (`LANE_LEFT`, `LANE_RIGHT`), adjust longitudinal speed (`FASTER`, `SLOWER`), or maintain its current motion (`IDLE`). This abstraction hides continuous vehicle dynamics, enabling efficient policy learning while preserving the key behavioral primitives needed to induce realistic interactions.

Reward Formulation

Our adversarial-training reward induces collisions while prioritizing behaviorally plausible, safety-relevant failures. It combines (i) a collision-likelihood shaping term based on the distance between the ego vehicle and the adversary and (ii) a terminal bonus awarded only when the resulting collision is classified as *valid* under the temporal-logic criteria in Section 6.2.1. Distance-based collision likelihood is commonly used in prior work and is adopted in approaches such as *DeepCollision* [29].

Safe-distance model Let v_{l} and v_{f} denote the longitudinal velocities of the leading and following vehicles, respectively. Both vehicles are in the same lane, with the adversary leading and the SUT following. The safe longitudinal distance $d^{\mathrm{lon}}_{\mathrm{safe}}$ is computed using a constant-deceleration model based on a Berkley algorithm [231], which is commonly used to compute safe distance [29]:

$$d^{\mathrm{lon}}_{\mathrm{safe}}(t) = \frac{1}{2}\left(\frac{v_{\mathrm{l}}(t)^2}{a_{\mathrm{l}}} \; - \; \frac{v_{\mathrm{f}}(t)^2}{a_{\mathrm{f}}}\right) \; + \; v_{\mathrm{f}}(t)\cdot\tau_{\mathrm{r}} \; + \; d^{\mathrm{lon}}_{\mathrm{min}}. \tag{6.4}$$

Here, a_{l} and a_{f} denote the maximum decelerations of the lead and following vehicles. In our experiments, we use the default values from `HighwayEnv`, setting both to $-5\,\mathrm{m/s^2}$. The minimum allowable longitudinal distance $d^{\mathrm{lon}}_{\mathrm{min}}$ is set to 5 m following *DeepCollision*. Unlike *DeepCollision*, which assumes $\tau_r = 0\,\mathrm{s}$ for an autonomous SUT, we set $\tau_r = 0.2\,\mathrm{s}$, motivated by reports indicating that expected ADS reaction times are on the order of 0.1 s [232].

When the vehicles are in different lanes, we use a safe lateral distance $d^{\mathrm{lat}}_{\mathrm{safe}}$, computed based on [233]:

$$d^{\mathrm{lat}}_{\mathrm{safe}}(t) = v_{lat}(t)\tau_r + \frac{1}{2}a^{\mathrm{lat}}_{\mathrm{max}}\tau_r^2 + \frac{\left(v_{lat}(t) + a_{lat}\tau_r\right)^2}{2b_{lat} + d^{\mathrm{lat}}_{\mathrm{min}}}. \tag{6.5}$$

Here, v_{lat} denotes the lateral velocity of the adversarial vehicle, τ_r is the reaction time, and a_{lat} and b_{lat} represent the lateral acceleration and deceleration, respectively. We set $\tau_r = 0.2\,\mathrm{s}$, $a_{\mathrm{lat}} = 5\,\mathrm{m/s^2}$, $b_{\mathrm{lat}} = -5\,\mathrm{m/s^2}$, and $d^{\mathrm{lat}}_{\mathrm{min}} = 2\,\mathrm{m}$.

At each timestep t, we compute the current distance between the vehicles as

$$d_c(t) = \sqrt{(x_{\mathrm{ego}}(t) - x_{\mathrm{adv}}(t))^2 + (y_{\mathrm{ego}}(t) - y_{\mathrm{adv}}(t))^2}. \tag{6.6}$$

The required safe distance is then

$$d_{\text{safe}}(t) = \begin{cases} d_{\text{safe}}^{\text{lon}}(t), & \text{if the vehicles are in the same lane,} \\ d_{\text{safe}}^{\text{lat}}(t), & \text{if the vehicles are in different lanes.} \end{cases}$$

Collision-likelihood shaping term Let $d_{\text{c}}(t)$ be the current distance between the two vehicles. We define the collision-likelihood reward as:

$$R_{\text{coll}}(t) = \begin{cases} \dfrac{d_{\text{safe}}(t) - d_{\text{c}}(t)}{d_{\text{safe}}}, & \text{if } d_{\text{c}}(t) < d_{\text{safe}}(t), \\ 0, & \text{otherwise.} \end{cases} \tag{6.7}$$

This term is assigned at each simulation step and increases as the adversary reduces the distance below the safe threshold, encouraging situations in which a collision becomes likely.

Validity-aware terminal bonus

$$R_{\text{valid}} = \begin{cases} 30, & \text{if failure is valid,} \\ -\dfrac{d_{\text{safe}}(t_m) - d_c(t_m)}{5}, & \text{otherwise.} \end{cases} \tag{6.8}$$

If the collision is invalid, we compute $d_{\text{safe}}(t_m)$ and $d_c(t_m)$ at the timestep t_m when the adversary initiated the unsafe maneuver and penalize violations of the safe-distance requirement. This term is applied once per episode, after a collision occurs.

Total reward and episode termination The full reward is

$$R = \sum_{t=0}^{T} R_{\text{coll}}(t) + R_{\text{valid}}. \tag{6.9}$$

The maximum episode duration is $T = 40$ steps. An episode terminates earlier at step T_c if a collision occurs. In this way, the adversary is encouraged not only to approach the ego vehicle dangerously, but to synthesize interactions that qualify as valid failures under our logical specifications.

State Representation.

The adversary observes the positions and velocities of both vehicles, as summarized in Table 6.1.

Table 6.1 State representation provided to the adversarial agent.

Vehicle	x	y	v_x	v_y
ego-vehicle	x_{ego}	y_{ego}	$v_{x,\text{ego}}$	$v_{y,\text{ego}}$
adversary	x_{adv}	y_{adv}	$v_{x,\text{adv}}$	$v_{y,\text{adv}}$

These features allow the policy to infer relative position, lane configuration, and closing speed, which are essential for planning lane changes and speed adjustments. The SUT observes the same state.

Adversarial RL training and Step 1 outcomes

We train the RL adversary using *DQN* [49] implemented in `CleanRL` [234]. We also evaluated *PPO* [51] and *SAC* [235], but *DQN* consistently achieved the strongest results, and we therefore adopt it in DYNASTO. Combined with the state representation, meta-action space, and reward formulation, this training yields an adversarial policy that reliably induces *valid* failures in the ego controller under the validity constraints of Section 6.2.1. Training produces a repertoire of failure-inducing behaviors for a fixed family of initial conditions. For each valid failure, we record the associated initial conditions and the adversary action at each timestep. These traces form the dynamic component that Step 2 replays while searching over initial scenario parameters to uncover additional failures driven by the interaction between static initialization and adversarial behavior.

6.2.3 Step 2: Initial Condition Search

In Step 2, DYNASTO targets the *static* initial conditions, such as vehicle positions, lanes, and speeds, which are fixed at episode start and lie outside the RL action space. Although Step 1 optimizes the dynamic action sequence, these parameters shape the resulting trajectory and determine whether replaying an adversarial behavior yields a valid failure. Step 2 therefore broadens the exploration beyond the relatively narrow initial-condition distribution induced by the default `highway` settings, complementing the stability-driven training regime of Step 1.

Black-box formulation and search objective

After an episode terminates, we treat the rollout as a black-box evaluation of a static initialization. For a given set of initial parameters, replaying the adversarial actions recorded in Step 1 yields a trace, and the failure-identification rules from Section 6.2.1 determine whether

it contains a valid failure. This motivates a metaheuristic search over static initial conditions, where each candidate is evaluated by a single rollout and the resulting outcome serves as a fitness signal. We adopt a GA because it is gradient-free, supports mixed continuous and discrete parameters, and can exploit episodic fitness derived from valid failures.

Search algorithm implementation

To search over the space of initial scenario parameters, we used the GA implementation from Pymoo library [179] with a continuous, normalized genotype representation. The GA evolves a population of candidate initial configurations, with each individual encoded as a vector in $[0, 1]^d$. The search objective is to discover initialization settings that induce failures when the failing scenarios recorded in Step 1 are rolled out.

Individual representation Each GA individual encodes a complete initial configuration of the ego and adversary vehicles. Let $\mathbf{x} \in [0, 1]^d$ be a genotype vector of dimension $d = 11$, corresponding to the parameters (position, current lane, heading, speed, target lane and id of selected scenario from Step 1) of both vehicles. The normalization is parameter-wise:

$$x_i = \frac{p_i - p_i^{\min}}{p_i^{\max} - p_i^{\min}},$$

where p_i is the raw scenario parameter and $(p_i^{\min}, p_i^{\max})$ are its admissible bounds. During phenotype reconstruction, the inverse transformation

$$p_i = p_i^{\min} + x_i(p_i^{\max} - p_i^{\min})$$

is applied, followed by rounding when p_i corresponds to a discrete parameter such as a lane identifier or scenario id.

This individual representation enables the GA to optimize continuous and discrete scenario attributes within a single real-valued search space.

Initial population The initial population is generated by independently sampling each parameter within its admissible range. For each dimension i, a value p_i is drawn from either a uniform real interval or an integer range, depending on its type. The resulting parameter vector is then normalized to $[0, 1]^d$ using the above-described transformation. All individuals share the same fixed-length encoding.

Crossover The GA uses Simulated Binary Crossover (SBX) [213] on pairs of parent vectors to generate offspring. SBX is well suited to real-valued encodings and mimics the behavior of single-point crossover on binary strings, producing offspring distributed around the parents according to a polynomial probability distribution. SBX ensures the exploitation of promising generated pairs.

Mutation After crossover, polynomial mutation (PM) [236] is applied independently to each dimension of an offspring vector. This operator perturbs each component with a probability given by the mutation rate, yielding small but effective local modifications while keeping all values within the $[0, 1]$ bounds. PM promotes the exploration of the parameter space while evolving the best ones.

Fitness function We rely on Equation 6.9 to evaluate the fitness of individuals, analogous to the reward calculation used for the adversary RL agent. This fitness value is assigned after the entire scenario has been executed and is multiplied by a negative sign, since the *Pymoo* framework supports only minimization objectives.

Duplicate removal We promote population diversity by discarding redundant individuals whose genotype matches one already present in the evaluated population. Using pairwise Euclidean distance with threshold D_{th}, two individuals are considered distinct only if they differ by at least 1.5% per dimension on average.

Simulation-based evaluation and Step 2 outcome

Unlike RL-based dynamic optimization, where agents interact with `HighwayEnv` at every simulation step, GA-based static optimization searches over initial conditions, with each individual representing a complete test configuration. Fitness evaluation therefore requires executing a full simulation episode. To integrate this process into the existing RL scenario-execution pipeline, we use an *ask-tell* interface. In the *ask* phase, the GA outputs a candidate test (a vector of initial positions, headings, speeds, and lane indices) without running any simulation. The candidate is injected into the environment and rolled out using the SUT policy and the recorded adversary actions. After termination, the resulting fitness is returned to the GA in the *tell* phase, and the process repeats for each individual across generations. This evaluation loop drives Step 2. Each candidate is decoded into a concrete initialization and paired with a Step 1 scenario selected via the decoded id. The recorded adversary actions provide the dynamic component, while the SUT continues to act online, as in Step 1.

The resulting trace is assessed using the failure-identification rules from Section 6.2.1, and configurations that trigger valid failures receive higher fitness and are preferentially propagated. Consequently, Step 2 expands the failure set beyond those seen during RL training by systematically varying static initial parameters while replaying failure-inducing behaviors. Step 2 is agnostic to how Step 1 produces failures; we use an RL-based adversary because it outperformed baselines in our evaluation.

6.2.4 Failure De-duplication and Analysis

The two-step scenario generation can produce a large set of valid failures, many of which are redundant or minor variations of the same underlying behavioral pattern. We therefore perform post-hoc failure de-duplication and analysis to remove redundancy and to assess how different testing configurations explore the failure space and its diversity.

Failure De-duplication

During scenario generation, all discovered scenarios are retained in memory. Afterward, highly similar recorded failures are grouped and filtered as duplicates, so they are not counted as distinct, unique failures. To compare failures, we represent each scenario with a descriptive vector that concatenates the normalized relative positions and speeds of the ego vehicle and the adversary over the last eight steps before collision, yielding a 24-dimensional embedding. Similarity is measured using a Euclidean-distance threshold S_{th}, calibrated on a manually curated set of about 100 similar and distinct scenarios. This calibration set is included in the replication package.

Failure Mode Clustering

As illustrated in Figure 6.3, we apply a graph-based clustering pipeline that groups similar failure traces into coherent failure modes using the next steps.

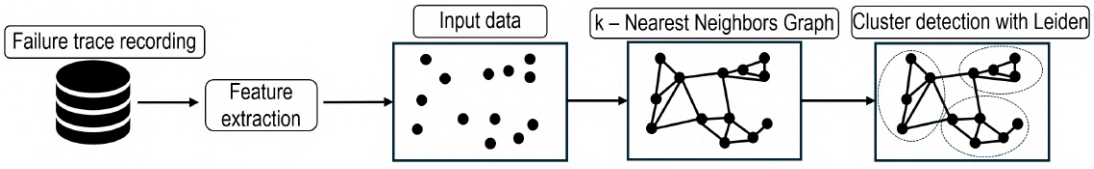


Figure 6.3 Failure cluster detection approach

Failure trace recording After a valid failure is identified, we record its full failure trace, including the time series of system states and the actions of both the adversarial and ego agents. We then post-process this trace using STL rules to detect events such as cut-ins, cut-outs, and braking by either the adversary or the ego vehicle. To support this event detection, the STL rules introduced in Section 6.2.1 are extended accordingly.

Feature extraction In Section 6.2.4, we introduce *descriptive vectors* that extract relative positions and speeds from each scenario trace. While these features capture low-level geometric and kinematic properties of the interaction, they often lack semantic structure and can be sensitive to minor trajectory variations. As a result, failures that are similar at a higher level, e.g., two cut-in scenarios with slightly different lateral offsets, may appear distinct under a purely low-level representation. For clustering analysis, we therefore extract higher-level features with clearer semantic meaning.
For each failure trace, we extract a sequence of high-level semantic events that characterizes how the interaction between the ego vehicle and the adversarial vehicle evolves over time. We use discrete behavioral descriptors such as lane-change patterns and braking interactions. To detect these events, we complement the STL rules from Section 6.2.1 (used to identify valid failures) with the following predicates:

$$
\begin{aligned}
Side_{adv} &:= x_{\text{ego}}(t) - V_L < x_{\text{adv}}(t) < x_{\text{ego}}(t) + V_L;\\
Side_{ego} &:= x_{\text{adv}}(t) - V_L < x_{\text{ego}}(t) < x_{\text{adv}}(t) + V_L;\\
Brake_{ego} &:= a_{\text{ego}} \leq -a_{\min};\\
Accel_{ego} &:= a_{\text{ego}} \geq a_{\min};\\
LaneC_{ego} &:= \big(\text{prev}(\text{lane}_{ego}) \neq \text{lane}_{ego}\big) \wedge \big(\text{prev}(\text{lane}_{adv}) = \text{lane}_{adv}\big);\\
CutIn_{ego} &:= \Diamond_{[t,t+\Delta t]}\Big(LaneC_{ego} \wedge Ahead_{ego} \wedge SameL\Big);\\
CutOut_{adv} &:= \Diamond_{[t,t+\Delta t]}\Big(LaneC_{adv} \wedge Ahead_{adv} \wedge \neg SameL\Big);\\
CutOut_{ego} &:= \Diamond_{[t,t+\Delta t]}\Big(LaneC_{ego} \wedge Ahead_{ego} \wedge \neg SameL\Big);\\
CutInSide_{ego} &:= \Diamond_{[t,t+\Delta t]}\Big(LaneC_{ego} \wedge SameL \ \wedge Side_{ego}\Big);\\
CutInSide_{adv} &:= \Diamond_{[t,t+\Delta t]}\Big(\text{prev}\big(LaneC_{adv}\big) \ \wedge SameL \wedge Side_{adv}\Big);\\
BrakeSL_{adv} &:= \Diamond_{[t,t+\Delta t]}\big(Ahead_{adv} \wedge Brake_{adv} \wedge SameL\big);\\
BrakeDL_{adv} &:= \Diamond_{[t,t+\Delta t]}\big(Ahead_{adv} \wedge Brake_{adv} \wedge \neg SameL.\big)
\end{aligned}
$$

Here $Side_{adv}$ and $Side_{ego}$ identify side-by-side configurations, where V_L is the vehicle length. These conditions are needed to identify side cut-ins. The predicates $Brake_{ego}$ and $Accel_{ego}$ capture braking and acceleration events of the ego vehicle based on a minimum acceleration

threshold $a_{\min}$. Lane-change events are encoded using $LaneC_{ego}$, which evaluates whether the ego vehicle has switched lanes relative to the previous timestep while the adversary remained in its lane. $CutIn_{ego}$ identifies an ego cut-in maneuver, while $CutInSide_{ego}$ and $CutInSide_{adv}$ detect lateral cut-in events in which a lane change leads to a side-by-side configuration. $CutOut_{adv}$ characterizes an adversary cut-out, where the adversary changes lanes and moves outside the ego's lane. Finally, $BrakeSL_{adv}$ and $BrakeDL_{adv}$ describe adversary braking events in the same lane or a different lane, respectively.
These predicates are evaluated at each simulation step and mapped to an integer identifier according to the following taxonomy:

0: none of the events below occurred;

1: `CutInSideEgo`, the ego changes lane while parallel to the adversary and collides with it;

2: `CutInEgo`, the ego cuts in ahead of the adversary;

3: `CutOutEgo`, the ego cuts out in front of the adversary;

4: `CutInSideAdv`, the adversary changes lane while parallel to the ego and collides with it;

5: `CutOutAdv`, the adversary cuts out of the lane ahead of the ego;

6: `CutInAdv`, the adversary cuts in ahead of the ego;

7: `BrakeSameLaneAdv`, the adversary slows down in front of the ego in the same lane;

8: `BrakeDiffentLaneAdv`, the adversary slows down in front of the ego in a different lane;

999: collision event.

Each failure trace is then represented as a time-ordered vector of event identifiers, which we refer to as an *event vector*. For example, the vector $[0, 0, 0, 6, 0, 0, 7, 7, 999]$ denotes a sequence in which the adversary first performs a cut-in maneuver, then slows down in the same lane, and ultimately a collision occurs. If multiple events occur at the same time step, we record their sum. To compare event sequences, we use the Levenshtein distance [237], which measures the minimum number of edit operations (insertions, deletions, and substitutions) required to transform one sequence into another. Thus, similarity depends on how many high-level events differ rather than on the absolute numerical values of the identifiers.

Construction of the k–nearest neighbors graph Given the matrix of event vectors, we compute pairwise distances using the selected distance metric (Levenshtein distance). For each failure trace, we identify its k most similar neighbors and connect them with undirected

edges, forming a k–nearest neighbors (kNN) graph. This graph captures the local similarity structure of the failure space: highly similar traces become densely connected, whereas outliers form sparse local structures. We assign edge weights proportional to similarity (i.e., $w = 1/(1 + d)$, where d is the distance between two traces), which guides the subsequent step.

Cluster detection with Leiden To automatically group similar failure traces into coherent modes, we apply the Leiden community-detection algorithm [227] to the weighted kNN graph. Leiden optimizes the modularity objective [238], favoring partitions with dense within-community connectivity and sparse connectivity across communities, and typically yields stable, well-separated clusters. Under this view, a failure mode corresponds to a community of traces that are strongly connected to each other but only weakly connected to the rest of the graph, capturing a shared underlying behavioral pattern. Our main motivation for using a community-detection algorithm is twofold. First, Leiden assigns every point to a cluster and does not rely solely on distance thresholds. Instead, it leverages the connectivity structure of the kNN graph to extract well-connected communities, whereas density-based methods often yield many outliers that require manual inspection. Second, Leiden is simple to tune, requiring only a small set of parameters, mainly the number of nearest neighbors (NN) and the distance metric. This makes it more practical than alternatives such as DBSCAN, HDBSCAN, or hierarchical clustering, which are commonly used in the testing literature but can be highly sensitive to hyperparameter choices [239]. In our cases, we selected the number of nearest neighbors (NN) using a labeled set of about 100 scenarios organized into seven clusters. We then tuned the distance metric so that the clusters produced by Leiden align with this reference set.

Clustering analysis outcome The resulting clusters correspond to distinct categories of failure modes (e.g., cut-ins, late braking, lane-change collisions). This clustering facilitates reasoning about the coverage and diversity of the revealed failures, supports the identification of dominant patterns, and highlights rare but safety-critical behaviors. It also provides a structured basis for comparing different testing configurations in terms of the failure modes they expose.

6.3 Evaluation

In this section, we first describe the experimental setup, then report quantitative and qualitative results.

6.3.1 Experimental setup

Below is our experimental setup for empirical evaluation.

Evaluation budget and statistical protocol. For each evaluated method, we allocate a total budget of 4000 steps. For DYNASTO, we assign 3000 steps to RL training in Step 1 and 1000 steps to the initial-condition search in Step 2. We repeat each evaluation at least 10 times. We report the non-parametric Mann-Whitney U test with significance level $\alpha = 0.05$, along with Cliff's delta effect size [182, 183]. The results of the statistical testing are illustrated in the boxplots. Pairwise statistical comparisons between configurations are indicated above the boxplots using horizontal brackets. Statistical significance is denoted using a star-based notation, where "**" indicates a statistically significant difference ($p < 0.01$) and "*" indicates ($p < 0.05$). Effect sizes are reported using dot notation, where ".." denotes a large effect and "." denotes a medium effect size.

Compute. All experiments are run on a PC with a 16-core AMD Ryzen 7 4800HS CPU @ 2.90 GHz, 16 GB of memory, and an NVidia GeForce GTX 1660 GPU with 6 GB.

Step 1 configuration. The RL adversary is trained with learning rate 5×10^{-4}, batch size 64, $\gamma = 0.95$, and a replay buffer of 150,000, using ε-greedy exploration with ε decaying from 1.0 to 0.05 over the first 20% of training. Initial conditions use two lanes, speed 25 m/s, heading 0 rad, with $x_{\text{ego}}! \sim [247, 263]$ and $x_{\text{adv}}! \sim [295, 327]$.

Step 2 configuration. We use a GA with population size 100 and crossover and mutation probabilities of 0.9. Parameters are sampled from Table 6.2 and include $(x_{\text{ego}}, x_{\text{adv}})$, $(h_{\text{ego}}, h_{\text{adv}})$, $(l_{\text{ego}}, l_{\text{adv}})$, and $(tl_{\text{ego}}, tl_{\text{adv}})$ (0 and 1 denote left and right lanes), plus a failure-id variable indexing recorded traces. We set the duplicate-removal threshold to $D_{th} = 0.05$.

Table 6.2 Ranges of initial parameters for Ego and Adversary vehicles.

Value	x_{ego}	x_{adv}	l_{ego}	l_{adv}	tl_{ego}	tl_{adv}	h_{ego}	h_{adv}	s_{ego}	s_{adv}
Min	247	395	0	0	0	0	-0.08	-0.08	20	20
Max	304	364	1	1	1	1	0.08	0.08	29	29

Systems under test. We evaluate two ego controllers trained with DQN in the `highway` environment of `HighwayEnv`, reusing the SUTs from Doreste et al. [30] for transferability and baseline comparison. Surrounding vehicles follow IDM policies [240]: SUT1 is trained with one standard IDM vehicle, and SUT2 with four defensive IDM vehicles. Both are trained for 8000 steps with rewards for high speed, collision avoidance, and staying in the right lane [30].

6.3.2 Results

Below, we report results organized in 4 research questions.

RQ1 – Effectiveness of Dynasto against Baselines

Motivation. In this RQ, we evaluate the effectiveness of DYNASTO against common adversarial test-generation baselines using the number of *valid, unique* failures uncovered under a fixed testing budget.
Method. We compare DYNASTO against four baselines on the two systems under test, SUT1 (UC1) and SUT2 (UC2):

- *Random Search (RS)*: at each step, the adversarial action is sampled uniformly from the available action set.
- *Genetic Algorithm (GA)*: GA precomputes a fixed adversarial action sequence before execution, yielding open-loop behavior that cannot adapt online to the ego.
- *Baseline (Base) (Doreste et al. [30])*: RL adversary trained with Deep Q-Learning that selects actions online but rewards all failures, without separating valid from invalid ones.
- *Validity-Aware RL (VARL)*: our RL adversary, whose reward explicitly promotes only valid failures under the temporal-logic criteria in Section 6.2.1.

Results. Figures 6.4 and 6.5 report the cumulative number of the revealed valid failures out of 4000 evaluations budget. For both SUTs, GA consistently outperforms RS with a large effect size, confirming that guided search is more effective than naive exploration. GA also surpasses the Baseline [30], despite its dynamic action selection, likely because the Baseline reward promotes all failures and is therefore less targeted for valid ones. Our validity-aware RL (VARL) adversary further improves over GA, with medium to large effect sizes on both SUTs, showing the benefit of context-dependent action selection with validity-aware rewards. Adding evolutionary search over initial conditions yields an additional gain: DYNASTO finds up to 70% more valid failures than DQN under the same budget, as reflected in the final failure counts (Figures 6.4 and 6.5).

Conclusions. DYNASTO augments RL with a GA over initial conditions, uncovering up to 70% more valid failures than the validity-aware DQN, which already outperforms the Baseline [30] under the same budget.

RQ2 – Dynasto Variants and Configurations

Motivation. In RQ2, we compare two integrations of RL with initial-condition search: *co-*

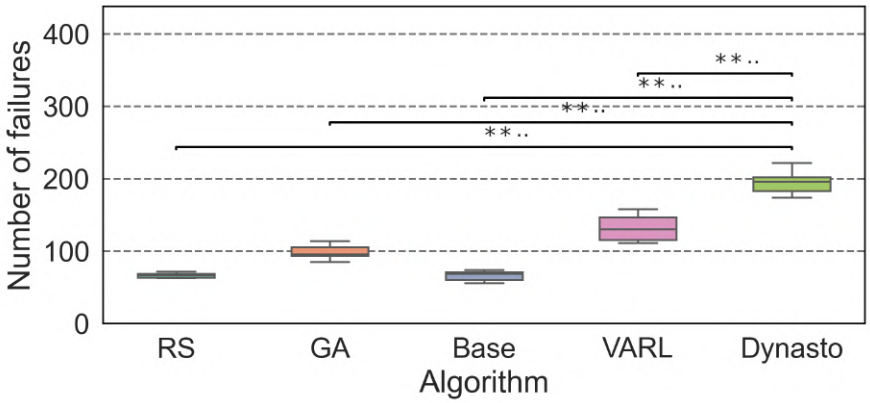


Figure 6.4 Number of failures revealed by algorithm in UC1

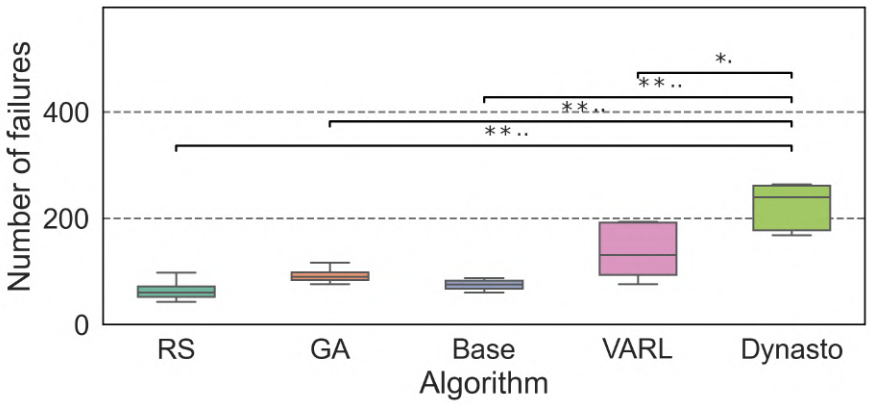


Figure 6.5 Number of failures revealed by algorithm in UC2

evolution (policy and initializations optimized jointly) and *sequential* DYNASTO (learn RL first, then replay traces to guide GA search).

Method. We compare five approaches across both SUTs:

- *VARL*: an adversarial validity-aware RL agent with no initial-condition search.
- *VARL+GA*: co-evolution of RL with GA, same configuration as DYNASTO(GA), except the chromosome omits the failure-trace selection gene.
- *VARL+RS*: co-evolution of RL with Random.
- *DYNASTO(GA)*: our proposed DYNASTO.
- *DYNASTO(RS)*: DYNASTO with Random instead of GA.

Results. Figures 6.6–6.7 report the cumulative count of discovered failures for SUT1 and SUT2, respectively. The statistical significance of the observed differences is summarized in the boxplots.

For SUT1, adding initial-condition search to RL improves over the *VARL* baseline in both co-evolutionary and sequential setups, with large effect sizes. Evolutionary variants (*VARL+GA* and *DYNASTO(GA)*) also outperform their random-search counterparts (*VARL+RS* and *DYNASTO(RS)*) with large effect sizes. Overall, *DYNASTO(GA)* is the strongest configuration, uncovering 11.4% more failures than co-evolutionary *VARL+GA*, with a medium effect size. For SUT2, only *DYNASTO(GA)* improves over *VARL* with a statistically significant, large effect size, revealing about 78% more failures than both the baseline and the co-evolutionary setups. Within DYNASTO variants, *DYNASTO(GA)* also outperforms *DYNASTO(RS)* with a medium effect size, yielding 31% more failures on average.

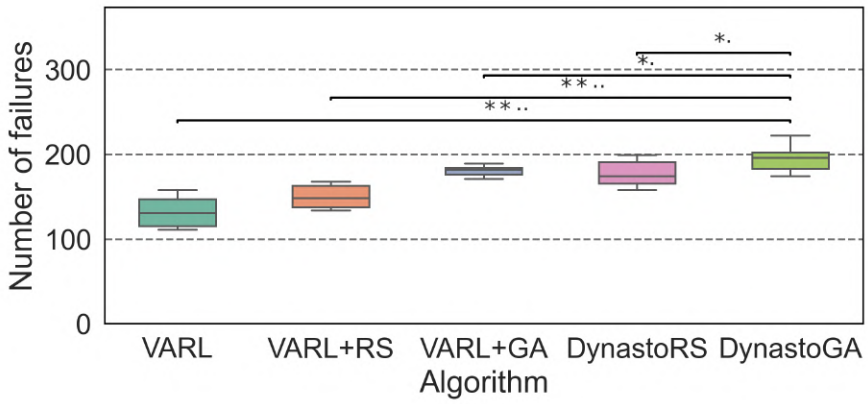


Figure 6.6 Revealed failures count by configuration for UC1

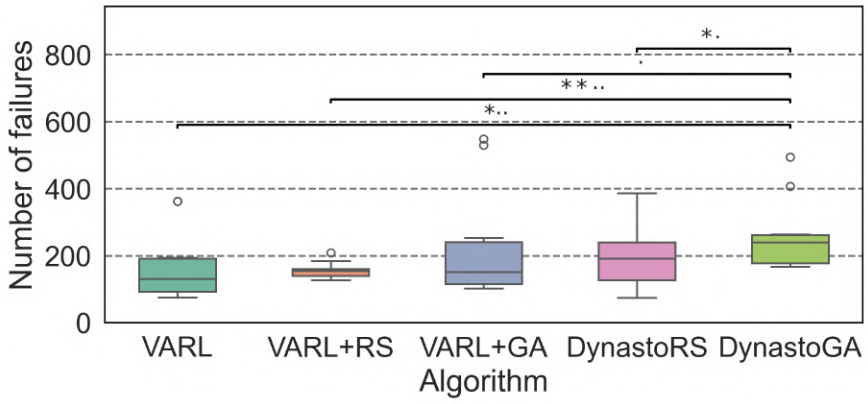


Figure 6.7 Revealed failures count by configuration for UC2

Conclusions. DYNASTO (*DynastoGA*) outperforms all baselines on both SUTs. The co-evolutionary setup (*VARL+GA*) can yield more failures in some runs but is less stable and more sensitive to initialization. DYNASTO is more consistent because it optimizes one component at a time.

RQ3 – Failure Analysis and Diversity

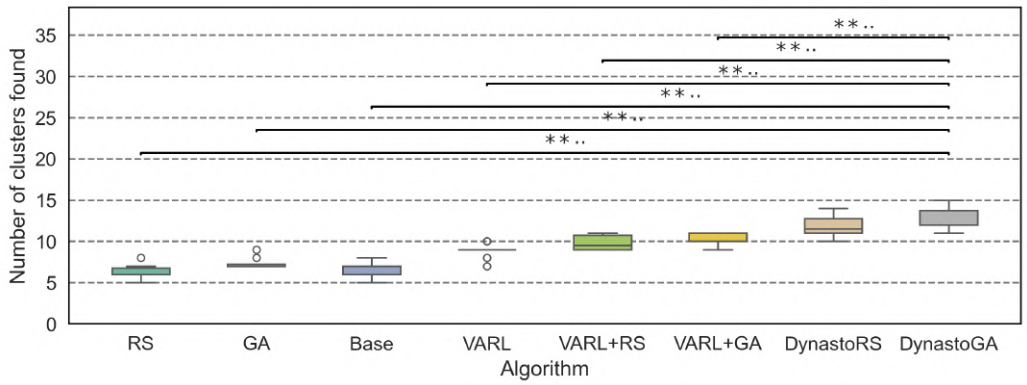


Figure 6.8 Failure clusters count discovered by method for UC1

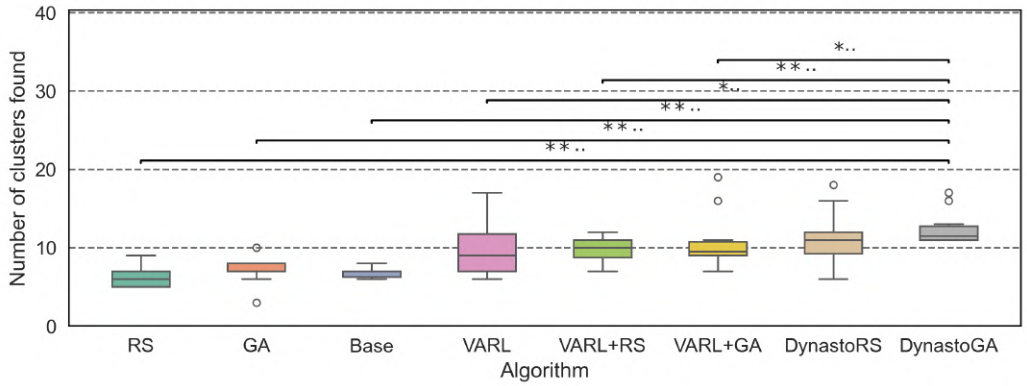
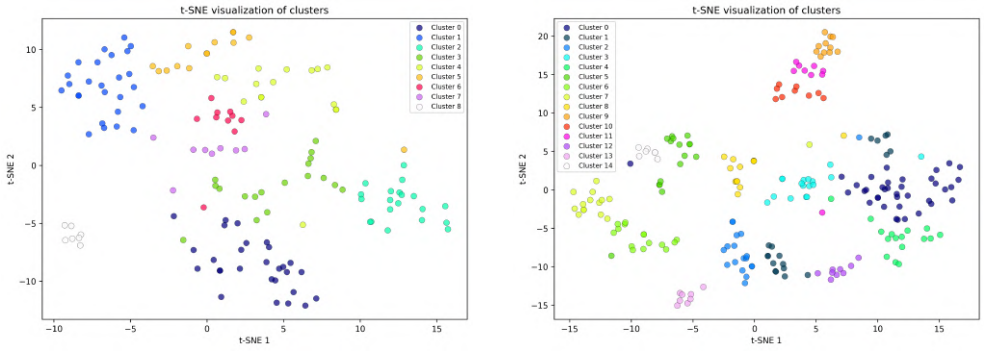


Figure 6.9 Failure clusters count discovered by method for UC2

Motivation. Unique-failure counts miss higher-level structure, so in this last RQ, we measure diversity by the number of failure clusters and characterize the dominant failure patterns.
Method. During scenario generation, we record all valid failures, remove duplicates, and cluster the remainder using the pipeline in Section 6.2.4. We then manually inspect the scenario videos within the clusters produced by DYNASTO(GA); all recordings are provided in the replication package.
Results. Figures 6.8–6.9 report the cumulative number of clusters for SUT1 and SUT2,

with statistical significance presented in the plots. The co-evolutionary setup that combines validity-aware RL (VARL) with initial-condition search improves over VARL without static-parameter search, revealing about 22% more clusters with a large effect size. DYNASTO outperforms all other methods with large effect sizes, uncovering up to 70% more clusters. Comparing search strategies, both GA and RS discover new patterns; GA yields slightly more clusters on average, but the difference is not statistically significant, indicating exploration comparable to random sampling despite GA's exploitation bias.

Figure 6.10 illustrates the clusters found by the baseline DQN and by DYNASTO for SUT1, and Figure 6.11 shows the corresponding clusters for SUT2. These visualizations indicate that DYNASTO discovers more failure clusters on average. We analyzed the failures contained in the DYNASTO-identified clusters for both systems under test. Below, we illustrate and describe selected failure scenarios. Overall, we observed similar failure patterns for both SUTs, with the exception of the failure pattern in Figure 6.16, which was observed only for SUT2.

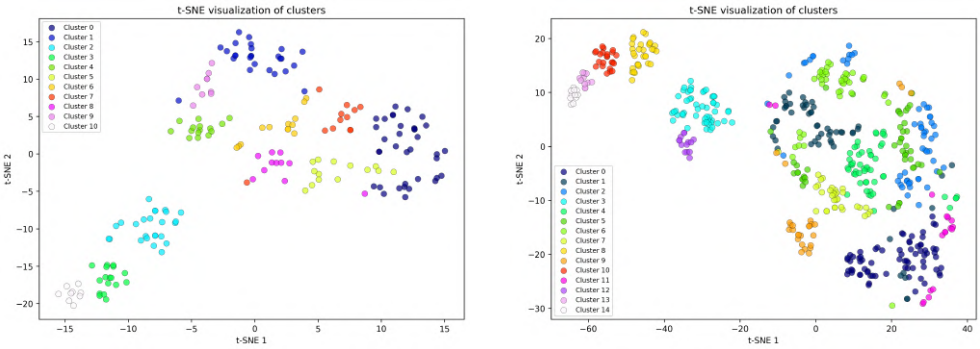


(a) Example of failure clusters discovered by DQN for SUT1

(b) Example of failure clusters discovered by Dynasto for SUT1

Figure 6.10 Failure clusters discovered by DQN and Dynasto for SUT 1

Next, we manually inspected representative clusters produced by DYNASTO for both SUTs. Figures 6.10 - 6.11 illustrate the discovered clusters.Overall, the dominant patterns fall into two categories: unsafe lateral maneuvers and inadequate longitudinal response. Specifically, we observe (i) unsafe lane changes and side collisions when the ego initiates a lateral maneuver with insufficient clearance to a nearby vehicle (Figure 6.14), (ii) rear-end collisions where the ego fails to brake when a lead vehicle slows down (Figure 6.13), and (iii) unsafe cut-ins where the ego merges with insufficient headway, causing the following vehicle to collide with its rear (Figures 6.12, 6.15, 6.16). We also observe a following-the-adversary pattern where the ego mirrors the adversary's lane change too closely and collides due to insufficient lateral



(a) Examples of failure clusters discovered by DQN for SUT2

(b) Example of failure clusters discovered by Dynasto for SUT2

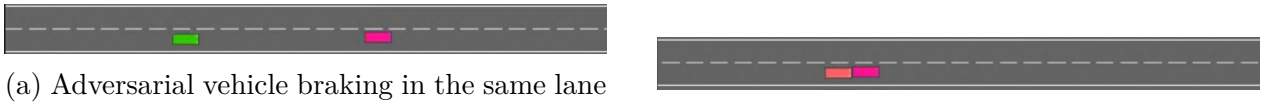
Figure 6.11 Failure clusters discovered by DQN and Dynasto for SUT2

separation. Finally, we identify one pattern unique to SUT2 in which the ego first adjusts its lane position to avoid the adversary, then performs an unsafe lane change while the adversary remains adjacent, leading to a lateral collision.

(a) Ego vehicle doing an unsafe lane change

(b) Collision with adversarial vehicle

Figure 6.12 Unsafe lane change failure pattern



(a) Adversarial vehicle braking in the same lane as the ego vehicle

(b) Ego vehicle failing to brake and adapt speed

Figure 6.13 Ego vehicle failing to brake

Conclusions. DYNASTO finds about 20% more clusters than the co-evolutionary setup and 50% more than the DQN baseline across both SUTs. Both GA- and random-search variants are effective, with GA yielding about 10% more clusters on average. Clustering also compresses roughly 200 unique DYNASTO failures into about 12 representative patterns per SUT, and manual inspection confirms these failures are valid and driven by unsafe ego behavior, revealing concrete weaknesses in decision-making and control.

(a) Ego vehicle performing an unsafe lane cut-in

(b) Collision with adversarial vehicle

Figure 6.14 Unsafe lane cut-in

(a) Adversary starting a lane change

(b) Ego vehicle performing the lane change after the adversary

(c) Ego vehicle colliding with the adversary

Figure 6.15 Ego vehicle following the adversary trajectory

(a) Ego vehicle avoiding the adversary

(b) Ego vehicle does an unsafe lane change

(c) Collision with the adversarial vehicle

Figure 6.16 Unsafe lane change failure pattern

6.3.3 Threats to Validity

We discuss threats to validity and mitigation measures.

Internal validity. All methods share the same well established simulator, action space, evaluation budget (4000 test executions), and validity criteria, but optimizers use different hyperparameters and convergence behavior, which can affect performance. We mitigate this by repeated runs with statistical testing and effect sizes and by reusing components (environment interface and STL-based labeling) across methods.

External validity. Our evaluation is limited to HighwayEnv enenvironment and 2-vehicle scenarios. To partially mitigate this threat, we conducted experiments with different systems under test.

Construct Validity. To mitigate threats to construct validity, we evaluate the proposed

metrics on controlled inputs. Specifically, we calibrate the STL-based validity rules using a manually annotated set of 50 scenarios, tune the clustering algorithm on datasets with known cluster structure, and select the duplicate-removal threshold using a curated set of similar and dissimilar test scenarios.

Conclusion Validity. Both reinforcement learning and genetic algorithms introduce stochasticity and run-to-run variability. We address this threat by repeating experiments multiple times, reporting statistical significance and effect sizes across runs.

6.3.4 Data Availability.

Replication package including the trained models, results, and implementation scripts are publicly available at: `https://figshare.com/s/2a193e5eeff51f9b5b27`

6.4 Chapter Summary

Testing autonomous driving systems requires uncovering not only many failures but also a diverse set of behaviorally plausible, safety-relevant failure modes. We proposed DYNASTO, a two-step scenario-generation approach that combines reinforcement learning for dynamic adversarial behavior generation with evolutionary search over static initial conditions. Operational knowledge of safety requirements is used to ensure the validity of the produced adversarial behaviors. Our empirical evaluation in *HighwayEnv* with two systems under test shows that a validity-aware RL adversary, used sequentially with GA search over initial conditions, improves failure discovery effectiveness, revealing over 22% more failures patterns compared to baselines. Future work includes extending DYNASTO to richer multi-agent ADS settings with higher-dimensional perception, refining notions of failure severity and coverage, evaluating the effectiveness of the approach and its failure validity awareness in more photorealistic, advanced autonomous driving simulators such as CARLA.

CHAPTER 7 IN-SIMULATION TESTING OF DEEP LEARNING VISION MODELS IN AUTONOMOUS ROBOTIC MANIPULATORS

7.1 Problem Context

Autonomous robotic manipulators (ARMs), also known as robotic arms, play a crucial role in industrial automation, being widely used for tasks such as palletization and machine tending. To perform tasks autonomously, these systems are equipped with various sensors, including cameras and lidars, allowing them to perceive their surroundings. Data from these sensors is processed by software incorporating machine learning components, such as deep neural networks (DNNs), which excel at detecting visual objects in 2D images, making them suitable for robotic guidance applications. However, the performance of these models is heavily dependent on the training data, which is typically collected and annotated manually, a process that is labor-intensive. Moreover, the collected samples may not encompass all edge cases that occur in foreseeable operational conditions. One solution to addressing the limited annotated datasets is the creation of synthetic data. This synthetic data can be initially used to evaluate the model's performance beyond the in-distribution samples. A selection of data points can then be incorporated into the next training set to enhance the model's robustness and generalization abilities.

On one hand, when synthetic data is generated through image transformations [241, 242] or generative models [243], there is no guarantee that it accurately represents naturally occurring use cases. These methods primarily introduce noise and distribution shifts to challenge the model's robustness against alterations in the input data. However, they often fail to produce operational corner cases that reflect specific, adverse environmental conditions. Consequently, while these techniques test the DNN's resilience to input variations, they may not adequately simulate the real-world complexities that the DNN will encounter in practical scenarios. Several methods [34, 145] leverage simulation environments to generate challenging and fault-revealing test inputs for the DNNs that are used to improve the DNN model for predictions in the real world. These works however, only consider the offline test data generation despite numerous studies [28, 244] highlighting the importance of conducting online testing for autonomous robotic systems with DNN-based components. Indeed, online evaluation of DNNs is important as statistical metrics like F1-score and mean Average Precision (mAP) do not allow to capture the failures arising from the DNN interactions with the environment and other system components.

Existing simulation-based testing methods are dedicated either to autonomous driving sys-

tems (ADS) [77, 81, 210] or unmanned aerial vehicles (UAVs) [85], with few studies focusing on ARM testing [245]. These methods often aim to expose weaknesses in DNNs by generating adversarial test cases through closed loop simulation of the system but do not propose solutions for repairing the identified failures. Therefore, we identify the need for test generation approaches that include both simulation-based online testing of autonomous robotic systems (ARS) with DNN components, as well as the system improvement from failures. Moreover, when improving the perception system from data collected in simulation, it is important to evaluate the usefulness of the improved system on the real world data.

To address the existing gap in the state-of-the-art approaches for DNN online testing and repair, we propose the MARTENS (Manipulator Robot Testing and Enhancement in Simulation) framework. This framework combines a photorealistic NVIDIA Isaac Sim simulator with evolutionary search for identifying critical scenarios, collecting useful data for system design assessment, testing and improving vision models. Using NVIDIA Isaac Sim, we created a photorealistic simulation environment for ARM applications, enabling the collection of synthetic 2D annotated images for training the DL vision models. The trained DL model is then deployed within the simulation to guide the robot in its manipulation tasks. MARTENS leverages evolutionary search to adjust structural parameters such as object positions and lighting, producing in-simulation tests that reveal failures. These failed test cases provide valuable data for refining the DL model and investigating the limitations of the control and perception strategies. Our evaluation, conducted through two industrial use cases involving pick-and-place tasks, demonstrates that our approach can effectively reveal diverse failures and can be leveraged to repair the DL model. On average, 31% of the generated in-simulation test cases resulted in the ARM failing the task. By fine-tuning the DL models with synthetic images from failed test cases, we achieved a 99% success rate in fixing these issues. A detailed analysis of the non-repaired test cases led to the discovery of some important robotic application design flaws. Finally, MARTENS facilitates the convergence to an optimal vision model using synthetic datasets, providing a pre-trained model that engineers can fine-tune with real-world datasets. An effective sim-to-real model transfer procedure reduces the need for extensive real-world data collection and labelling as well as online testing of the vision model using a real robotic system. We evaluate our model trained with synthetic data on the real world images, matching the scene represented in the simulation environment. The model trained and repaired using the MARTENS approach achieved mean average precision (mAP) scores of 0.91 and 0.82 on real-world images with no prior retraining. Further fine-tuning on real-world images for a few epochs (less than 10) increased the mAP to 0.95 and 0.89 for the first and second use cases (UC-1 and UC-2), respectively. In contrast, a model trained solely on real-world data achieved mAPs of 0.8 and 0.75 for UC-1 and UC-2 after more than 25

epochs.

7.2 Approach

The overall diagram of the MARTENS approach illustrating the major steps is shown in Figure 7.1. In this section, we provide a detailed overview of each of the indicated steps.

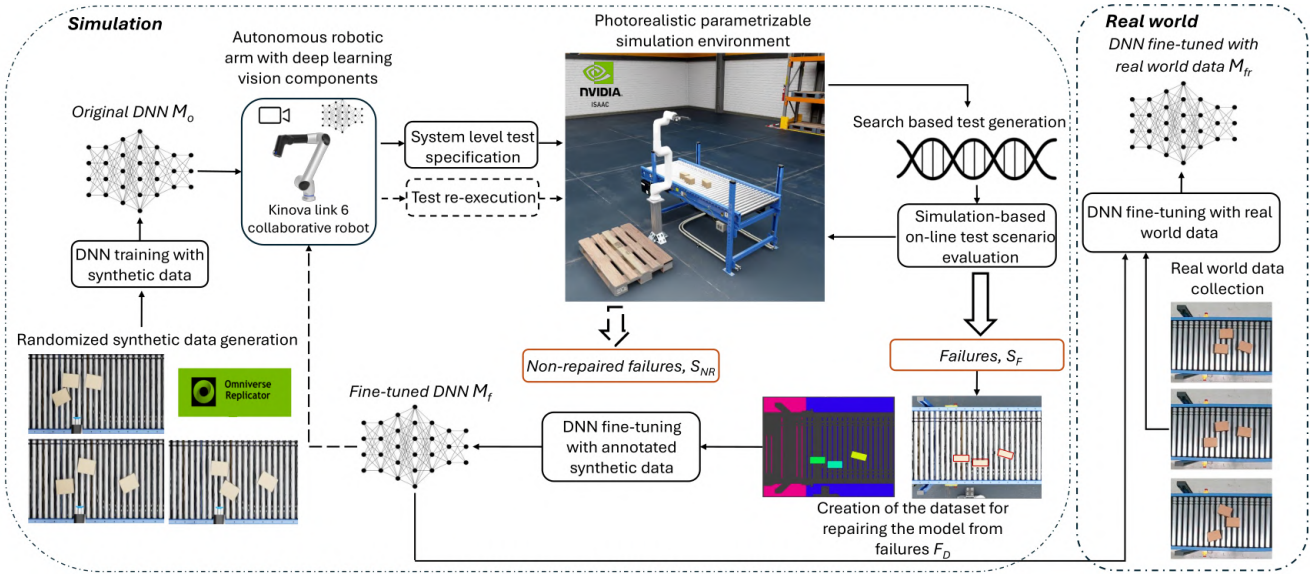


Figure 7.1 The diagram of the MARTENS approach

7.2.1 In-simulation Test Environment Specification

The first step in testing perception components in the simulation is setting up a photorealistic parametrizable environment. It should mimic the production environment and produce the variability needed for testing the vision system by randomizing parameters. In a typical environment for testing autonomous robotic manipulators, there are objects with fixed position (static objects) O_s, objects with variable position (flexible objects) O_f, a robot R, a light source L and a camera C with a preset angle of view. To create this typical environment within Isaac Sim, the robot R should be imported using its standard format, URDF (Unified Robot Description Format), which is generally provided by the manufacturer. The robot's URDF contains its physical description, including joints, mass, etc. The objects O_s and O_f can be imported to the simulation from their corresponding 3D computer aided design (CAD) models. The light sources and virtual cameras are included within Isaac Sim and can be conveniently configured. Indeed, Isaac Sim features different types of light sources [246]

such as rectangular light, distant light, sphere light, cylindrical light, etc. To replicate a conventional lighting condition, we can place the light source L higher in the ceiling and use the rectangular light type that emits a uniform illumination. Varying luminosity is important for testing the robustness of vision models [226].

(a) RGB image from the camera (b) Segmented image

Figure 7.2 Camera view in the virtual environment

The camera C has a pre-defined angle of view and provides RGB images of the virtual environment that are then processed by the vision system. The camera can be located in a fixed position in the scene, also known as 'eye-to-hand' camera or it can be mounted on the robot, known as 'eye-in-hand' camera. Figure 7.2a displays an image captured by an 'eye-to-hand' camera mounted on top of the manipulable objects. At inference time, the DL vision predicts the target outputs given the RGB images captured by the virtual camera. However, the estimated pixel coordinates within the 2D images should be projected into the world space of the robot. In real-world production settings, the projection matrix is generally computed using 3D camera calibration [10]. Isaac Sim API provides a function that allows to obtain the coordinates in the world frame from the pixel coordinates in the image. This eliminates the need to calibrate the camera in the simulation and allows for testing the DL vision model with an ideal camera calibration. Moreover, the simulator provides the ground truth information on the position and class of every object in the scene, as shown in Figure 7.2b.

The ARM R is tasked with a pick and place task of the flexible objects O_f. The ARM should satisfy two requirements while completing the task. The first requirement R_1 is that each placed O_{fi} should be placed parallel to the palette with no more than ϕ_d degrees of deviation. The second requirement is to perform the placement of each O_{fi} in the target location. Having set up the test environment in the simulator, we determine the environmental factors that could influence the vision model predictions and the outcomes of the vision-guided robot movements. Given the typical environment of an ARM application described above as well as the system performance requirements, we find that the location and orientation of the

flexible objects and the lighting intensity affect the vision model predictions. In order to perform the in-simulation testing, these parameters must be dynamically changed through simulation control logic. In the following, we discuss how these parameters are encoded into a vector to allow searching algorithms to explore the parameter space. The goal of the search is to find such set of parameters that make the ARM falsify the established requirements R_1 and/or R_2.

7.2.2 Search-based Test Generation

In simulation, randomizing environmental parameters can help generate valid test cases for evaluating vision-guided robotic systems. Nevertheless, the main purpose of testing is to reveal failures by which we can identify design flaws, bugs in the software, and DNN inefficiencies. There are many methods to guide the search of environmental parameters towards the failure-revealing regions. Previously, evolutionary algorithm-based approaches were proven effective at test generation for autonomous robotic systems [38, 247]. Below, we provide a more detailed description of the elements of our genetic algorithm (GA) based test generation approach.

Individual representation and sampling. We represent the individuals, or chromosomes, as one-dimensional arrays of size $3n + 1$. It contains the concatenation of the n flexible objects' positions, defined as sub-vectors of $[x_i, y_i, r_i]$, where x_i, y_i is the object position in the 3D space with the z-coordinate fixed at a pre-defined value and r_i is the rotation angle of the object w.r.t z-axis. The rotation angles w.r.t x and y-axis are fixed and set to 0. The last dimension corresponds to the luminosity l level in the scene. We define constraints to delimit the vector space in order to ensure that: (i) no intersection occurs between the objects; (ii) positions of objects are in the camera's field of view. Indeed, our use cases involve pick and place operations of objects on a planar surface with known height (the z-coordinate is constant or can be read from a depth sensor). Hence, (x, y) coordinates and rotation with respect to the z-axis are sufficient to represent the locations and orientations of objects O_f posed on a planar surface.

Fitness function. Fitness function quantifies how well an individual meets the objective, guiding the selection of superior individuals over the generations [40]. Robot movement success or failure is a binary information that leads to a sparse fitness function. A continuous score is more suitable for comparing individuals in a generation. Since the DL vision model detects the objects in the 2D image, we calculate the fitness based on the maximum deviation of the estimated object position by the vision system from the actual object position as well as the maximum deviation of the predicted rotation angle from its associated ground truth.

The fitness can be formulated as follows:

$$F = \frac{w_1}{k_p} \cdot \max_{n \in N} \| \ p_n - p'_n \| + \frac{w_2}{k_r} \cdot \max_{n \in N} \| r_n - r'_n \|, \tag{7.1}$$

where N is the total number of objects. Parameter p_n corresponds to the predicted position of the object and p_n' of the ground truth object position. r_n corresponds to the estimated object rotation, while the r_n' is the actual object rotation. The two deviations w.r.t object location and orientation are not conflicting objectives, so we can use a single-objective genetic algorithm [40], rather than a multi-objective one, such as NSGA-II [47]. The coefficients w_1 and w_2 are used to assign weight to objectives. The k_p and k_r coefficients normalize the values of the objectives to be from the same range.

Crossover operator. We are using a one-point crossover operator, which is one of the commonly-used operators [209]. Given a pair of parents the crossover point is randomly determined. At this point, genes (components of the chromosomes) are exchanged. Following the generation of the offspring, a constraint feasibility validation is performed. If the constraint is violated, a low fitness value is assigned to the individual.

Mutation operator. We use two mutation operators: random modification operator M_{rm} and replacement operator M_r. M_{rm} operator choose k arbitrary elements of the chromosome (i.e. genes) and then increases or decreases the value of this gene by 10 %. We vary k from 1 to the chromosome size. M_r operator randomly chooses one object described by a set of (x_i, y_i, r_i) values and replaces them with randomly sampled values. The intuition is to change the position and orientation of one of the objects. At the same time, it is ensured that new object position will not cause collisions with the existing boxes. At each generation, one of the operators M_r or M_{rm} is applied with the probability of p_{mut}.

Duplicate removal. At each iteration, we remove the individuals which have a small deviation between them. We estimate the deviation as the cosine distance D_c [211] and compare it to a prefixed minimum threshold, D_{cth}. If the D_c between the two individuals is less than D_{cth}, one of them is removed.

The described algorithm is used to generate the test cases that are then evaluated in the simulation environment. Tests are considered failed if the robotic system is unable to pick or place the objects properly. In simulation, the obtained system behaviors can be compared to their desired counterparts, and a maximum deviation threshold can be established to separate passed from failed tests. We define the thresholds to identify the system failures in our use cases in Section 8.3.1. As a baseline for the evolutionary search, we implement random search (RS) to generate test cases by arbitrarily selecting values from the allowable ranges. With

proper configuration, GA-based search should outperform RS in terms of the proportion of failures revealed while maintaining diversity among the detected failures. For both competing search strategies, the test generation budget is defined by either the allowable total execution time or the maximum number of evaluations. Moreover, the duplicate removal step is applied to both GA and RS.

7.2.3 Test Evaluation

Generally, we consider the test failed if at least one of the performance requirements is falsified i.e. the object is not placed correctly on the pallet (i.e., it is not aligned with the pallet, failing the requirement R_1) or it is not successfully picked and transported to the target position (failing requirement R_2). During the test generation, both passed and failed test cases are saved in the corresponding sets S_P and S_F. As we focus on investigating and repairing the test cases with DNN mispredictions, we further split the failures into two subcategories: (i) *soft* failures, where the DNN prediction was not accurate, leading to a violation of one of the requirements; (ii) *hard* failures, where the DNN prediction was acceptable, but the system failed to satisfy either the requirement R_1 or R_2. Afterwards, we analyse on characteristics of the failed tests S_F in terms of structural features sparseness (test diversity) and severity sparseness (failure mode diversity). Indeed, we leverage the failure sparseness, S_{av}, a metric proposed in the literature [81], to estimate the diversity by computing the average maximum value of the distance metric between each pair of the failed test cases *tc*:

$$S_{av} = \frac{\sum_i^N \max_j^N \text{dist}(tc_i, tc_j)}{N} \tag{7.2}$$

where $N = |S_F|$ is the total number of the failed test cases, and *dist* is the distance metric. We used the cosine distance metric D_C to compute structural features sparseness, S_{avf}. To evaluate the severity sparseness, we defined five possible failure modes of the system, FM, based on domain knowledge and requirements: incorrect box center predicted by DNN FM_1, incorrect box orientation predicted by DNN FM_2, failure to place the box FM_3, incorrect box orientation at placement FM_4 and robot stuck during the execution FM_5. For each test case, we collect all the failure modes observed during the pick-and-place of each flexible object, O_f. Then, to evaluate the diversity of failure modes, we use Eq. (7.2), but with a different distance metric *dist*. We used the number of the identified unique combinations of the failures modes N_{FM} to estimate the distance between two test cases, thereby computing the severity sparseness, S_{avs}. Intuitively, the more unique combination of failure modes are discovered, the better output behavior coverage is achieved.

7.2.4 Vision Model with Randomized Synthetic Data

As a simulation platform, we selected Nvidia Isaac Sim, which enables advanced GPU-enabled physics simulations with high photorealism. It includes Isaac Replicator [248] that relies on environmental parameters to provide image metadata information such as depth maps and 2D annotations in the form of color-based segmentation (see Figure7.2b). For pick-and-place applications using 2D vision, we post-process the segmented images to extract the contours of the objects of interest O_f, then, the polygon coordinates delimiting each object are derived to be added as groundtruth annotations with the RGB images. Using the Isaac Replicator API, we propose generating annotated synthetic images by randomizing environmental parameters within specified distributions. This approach produces scenes with diverse object positions and orientations (x_i, y_i and r_i), as well as varying luminosity levels l. The resulting synthetic dataset enables the vision model to learn and generalize, effectively, improving its ability to accurately detect and position objects under various conditions. Then, after training, the model's performance is evaluated using statistical metrics like mean Average Precision (mAP) based on synthetic validation data collected the same way as the training dataset. The average of mAP is calculated at varying intersection over union (IoU) thresholds, ranging from 0.50 to 0.95 This offline validation process mirrors how vision engineers assess their models' performance to optimize architecture and hyperparameters. However, achieving a perfect mAP of 1.0 is unlikely to be feasible and can even indicate overfitting. Therefore, testing the vision model in production, integrated with the robotic arm, becomes crucial to verify that the DNN's precision is adequate for ensuring successful picking of the object and placement with correct orientation. In the following sections, we discuss the online evaluation of the ARM with the vision model operating within the above-introduced flexible simulation environment. The proposed in-simulation testing serves to refine the vision model by fine-tuning it on examples with its mispredictions and failures, in addition to uncovering potential design flaws in the vision-based robotic arm system early on.

7.2.5 Model Repair Using Failure Data

The dataset for repairing the DNN model includes all the RGB images of the scene with objects O_f from the camera C, when the ARM failures were detected. For each image, the segmentation data is extracted from the simulator. After post-processing the segmented images, we obtain a dataset D_F of images and labels suitable for model training. This dataset includes all failures identified by the generated test cases, S_F, along with a selection of passed tests from S_P, where the system completed the task but the DNN predictions were inaccurate compared to the ground truth provided by Isaac Sim. Specifically, these are cases

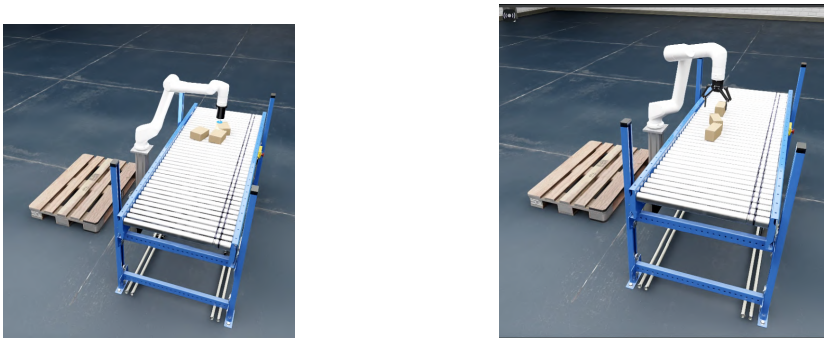
where the predicted orientation of the cardboard boxes deviated from the ground truth by more than 5 degrees, or where the predicted center of the boxes deviated by more than 1 cm. These test cases can be categorized as near-fails, denoted S_{NF}. Adding them to our training dataset for fine-tuning can help the model improve its inductive bias and enhance its performance, reducing the likelihood of such cases turning into actual failures. Thanks to transfer learning, the DL vision model does not need to be retrained from scratch on the entire dataset, i.e., combining the original and the novel datasets. Instead, the fine-tuning can be done directly on the last operating model, M_o, considering it as the base (pretrained) model, and fine-tuning it exclusively on the failed dataset, D_F, using the same hyperparameters, except for decreasing the maximum number of epochs. This reduction helps mitigating the risk of catastrophic forgetting phenomenon, where the model overfits to the new samples to the extent of losing its original performance. Furthermore, we use a merged validation dataset that includes original validation data and novel validation data (the collected failed test data). Even with high performance on validation set, offline assessment alone cannot ensure that most of the failures are resolved. It is essential to replay the revealed failures in a subsequent in-simulation testing session using the fine-tuned model M_f. Thus, we re-run the failed test cases from S_F and record a new set of test cases that passed after the model repair S_R ('repaired' test cases) and a set of test cases that did not pass S_{NR} after the repair ('non-repaired' test cases). As a result, we will be able to measure how many failures were fixed by DL vision model repair.

7.3 Evaluation

7.3.1 Experimental Setup

Use cases. We evaluate our approach on two test subjects that are autonomous robotic manipulators performing pick and place task as illustrated in Figure 7.3. The task aims to automate the process of pulling cardboxes from a conveyor belt and arranging them appropriately on a pallet for delivery. In this task, the robotic manipulators are used for streamlining packaging operations and ensuring efficient transport loading. For both test subjects we use a Kinova link 6 [249] collaborative robot. First test subject (test subject 1) is equipped with suction gripper, while the second subject (test subject 2) with a parallel gripper (Robotiq 2F-85 [250]). Test subject 1 tackles the task of pick and place cardboxes of the 17 x 14 cm size, while the test subject 2 handles the 12 x 8 cm cardboxes. Parallel grippers typically require more precise positioning and control to ensure a successful grasp, than the suction grippers [251]. Both systems have a fixed camera mounted on top of the conveyor belt's region of interest, i.e., reachable by the robot. The camera is an Intel RealSense

D435 [252], configured to capture images of the size 640 x 514 pixels.

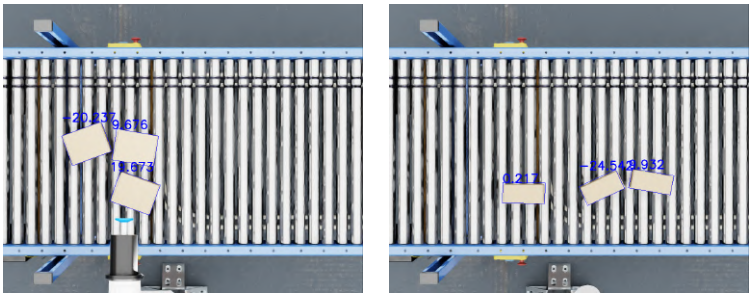


(a) Manipulator with suction gripper (b) Manipulator with parallel gripper

Figure 7.3 Illustrations of the test subjects used in the study

The robot needs accurate vision guidance to achieve the grasping of the objects and then complete their proper placement in the pre-defined locations (i.e., properly placed means correct position and orientation). The robot starts in the default position, also known as 'home' position. Then, the image is captured with the fixed camera. If cardboxes are detected by the DL model in the image, the controller initiates the pick and place cycle of the box with the highest detection confidence. The box center coordinates are translated into the robot's coordinate frame and the rotation angle w.r.t the z-axis is sent to the robot to grasp the cardbox with the correct orientation. Once the box is securely grasped, the robot lifts it off the conveyor belt and transports it to the place position. Then, the gripper is opened, box is placed and the robot returns to the home position, where the cycle repeats. When no cardboxes are detected, the controller is stopped. Typically, this cycle is repeated for 3 times as we fixed the number of cadboxes to 3. To implement robot control within Isaac Sim, we used the RMPFlow controller [253], which encapsulates an inverse kinematics solver and enables the implementation of robot primitives for the necessary movements based on computed and vision-estimated positions. When the actual robotic system is implemented, these primitives will be handled by Kinova's Kortex API [254]. In both use cases, two operational requirements are defined as follows: R_1 - place the object with a 0-degree orientation relative to the pallet and R_2 - perform the pick and transport the object to the target position. The failure is detected when the box orientation at the place location deviates by more than 5 degrees from the target. Failure is also detected if the box is positioned more than 50% of its size away from the target location.

Vision models. The vision models are responsible for detecting the positions and orientations of the target objects i.e. cardboxes. An example of the model output predictions are



(a) Test subject 1 view (b) Test subject 2 view

Figure 7.4 Top views from the fixed camera for the subjects under test with model predictions

shown in Figure 8.3. Indeed, the vision system should send the coordinates of the cardbox center and its rotation angle w.r.t the z-axis. This information can be inferred from the 2D images using YOLOv8 [255] that natively supports the oriented bounding box detection. As explained in Section 7.2.4, images and coordinates of polygons defining cardboxes will be prepared. Such labelled datasets can be used to train YOLOv8 for oriented bounding box detection [256]: its center location, height, width as well as its orientation. For each of the use cases, we trained a dedicated vision DL model using 1200 synthetic images. The images were collected by randomizing the scene parameters through the Isaac Replicator API following the uniform sampling from the ranges shown in Table 8.1 with the orientation of the boxes ranging from -25 to 25 degrees. We then split the dataset into train set of 960 images and a validation set of 240 images. As recommended by Yolov8 creators [257], we used a batch size of 16, an adaptive learning rate from 0.01 to 0.0001, and image augmentations like changing the brightness, rotation, scaling, flipping and shear. Using early-stopping on validation data, we trained the models for 100 epochs and both best-fitted models achieved a mean average precision (mAP) of 0.98. Having such high precision is expected since the synthetic dataset is collected using a simulation environment and a uniform sampled data from the same distribution. Online in-simulation testing of the model aims to find inputs that reduce its precision in identifying objects in the scene.

To run the experiments we used ‘g5.2xlarge’ [258] AWS instance with the following characteristics: A10G Tensor Core GPU with 24 Gb of memory, 8 AMD EPYC CPUs, 32 Gb RAM. In comparing the results obtained from the randomized algorithms, we repeated all evaluations at least 5 times due to the computational cost of running the simulations. For all results, we report the non-parametric permutation test [259] with a significance level of $\alpha = 0.05$, as well as the effect size measure in terms of Cliff’s delta [183]. Given the computational constraints to obtain more runs of the algorithms, we chose permutation test as it

makes fewer assumptions about the input data compared to other non-parametric tests such as Mann-Whitney U test [182].

Search algorithm configuration. For both use cases, we fixed the number of flexible objects O_f, i.e. cardboxes, to $N = 3$. The chromosomes are 10-D arrays with the first 9 dimensions representing the positions (x_i, y_i) and rotations (r_i) around the z-axis of the three cardboxes. The last dimension represents the luminosity level l at the scene. The initial population is generated by randomly sampling the values in the allowed ranges, as specified in Table 8.1. We calculated the fitness function using Eq. (7.1), with equal weight

Table 7.1 Allowed ranges for the parameters

x_i	y_i	r_i	l
$0.5 - 0.9$	$0 - 0.92$	$-30 - 30$	$1500 - 5000$

coefficients w_1 and w_2 set to 0.5. The normalization coefficients k_p and k_r were chosen based on previously recorded values and set to 0.01 and 1, respectively. We fix the value of mutation probability p_{mut} as 0.4, crossover rate p_{cross} as 0.9 and the duplicate removal threshold D_{cth} as 0.1. These values are influenced by our previous studies [157] and confirmed in the warm-up experiments. Population size was set to 40 with the execution budget of 220 evaluations. Over the generations, each individual is used to create the test case scene and evaluate the simulated robotic arm in performing the pick and place of the three cardboxes. The evaluation of each test scene using Isaac Sim is computationally expensive and takes on average 280 seconds for use case 1 (UC-1) and 310 seconds for use case 2 (UC-2) using the GPU-powered AWS instance: 'g5.2xlarge'. For failure modes, they can be identified as follows: FM_1 occurs when the center O_f predicted by the DNN model deviates from the ground truth by more than 1 cm. FM_2 is noted when the predicted orientation of O_f deviates by more than 5 degrees from the ground truth. FM_3 indicates that O_f was not placed in the target location. FM_4 occurs when the orientation of the placed O_f deviates by more than 5 degrees from the target orientation. FM_5 denotes when the ARM gets stuck during execution and restarts the operation cycle.

Expert feedback. Given that this research was conducted in partnership with industrial collaborators from Sycodal, we had the opportunity to gather domain expert feedback on each of our research questions (RQs). We conducted two 1-hour interviews with two robotic and vision engineers at Sycodal. During these interviews, we first presented the results obtained for each RQ, followed by a 15-minute discussion. We noted all the points mentioned by the engineers and present them at the end of each RQ.

7.3.2 Research Questions

RQ1: Effectiveness of MARTENS in revealing ARM failures

Motivation. We aim to ensure that the designed in-simulation testing method is able to reveal system failures. Our goal is also to determine whether evolutionary search (GA) can reveal more diverse and numerous failures than random search (RS).

Method. For both use cases, we perform five runs of RS and GA with 220 evaluations each. Then, we compute the number of passed versus failed test cases, as well as the near-failed test cases. To distinguish the failure involving DNN prediction inaccuracy and the system failures even with correct predictions from DNN, we calculate the number of soft versus hard failures. Understanding the failures that occurred when the DNN predictions were adequate is critical as it can point towards some design flows on the control algorithm level. To compare between the GA and RS beyond the number of revealed failures, we measure the structural features sparseness among test environments generated by each method, as well as, the severity sparseness. This gives us the distinctive characteristics among the revealed failures in terms of diversity and coverage.

Table 7.2 Generated tests statistics for UC-1

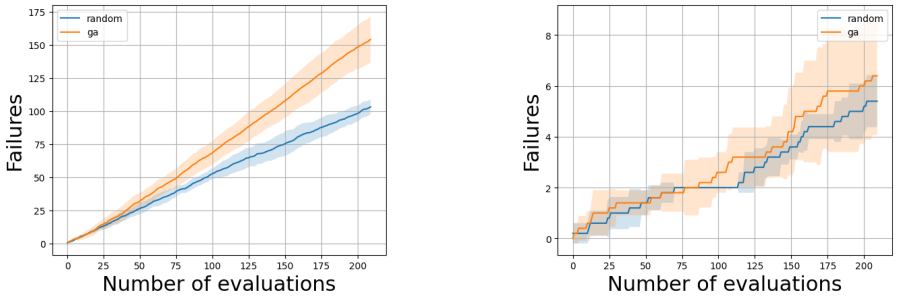
Algorithm	**All Tests**			**Failure Type**		**Total**
	Pass	**Fail**	**Near Fail**	**Soft**	**Hard**	
RS	531	532	26	510	22	1093
GA	243	794	45	782	12	1082
Total	774	1326	71	1292	34	2175

Table 7.3 Generated test statistics for UC-2

Algorithm	**All Tests**			**Failure Type**		**Total**
	Pass	**Fail**	**Near Fail**	**Soft**	**Hard**	
RS	1038	27	33	9	18	1100
GA	935	34	112	12	22	1084
Total	1973	61	145	21	40	2184

Results. Table 7.2 and Table 7.3 present all the statistics of passed and failed test cases, as well as their sub-categories for UC-1 and UC-2, respectively. Overall, our approach reveals 1326 and 61 failures in UC-1 and UC-2, respectively, based on more than 2175 generated test cases. Examples of the revealed failing test cases with DNN mispredictions are available online [260]. For the UC-1, the majority of the failures occurred due to model inefficiencies,

which affects the precision of vision-based picks and results in 97.4% soft failures. For the UC-2, 65.5 % of the failures are hard, meaning they happened even when the DNN model predicted correctly, suggesting that the pose estimation accuracy needs to be increased or potential design flaws exist. When comparing the numbers obtained by using each searching strategy, we find that GA-driven test generation reveals 50 % (UC-1) and 26 % (UC-2) more failures, than RS-driven test generation. Interestingly, in UC-1 RS reveals more hard failures than RS. We attribute it to the fact that the fitness function is designed to directly promote DNN mispredictions i.e soft failures, rather than the task level failures. Our GA implementation successfully evolves the generations towards being more likely to exploit system vulnerabilities and cause system failures. Figure 7.5 illustrates the evolution of the number of failures revealed by each strategy over the iterations of the search. Initially, the GA-based test generation begins with random test cases, displaying a similar likelihood of inducing failures as the RS approach. However, over successive iterations, the GA method increasingly exposes more failures compared to RS. This widening gap demonstrates that the GA-based generations are progressively improving. In the following analysis, we compare the structural features and severity sparseness among the test cases generated by GA and RS to evaluate them in terms of test diversity and failure mode coverage.



(a) Failure convergence for use case 1 (b) Failure convergence for use case 2

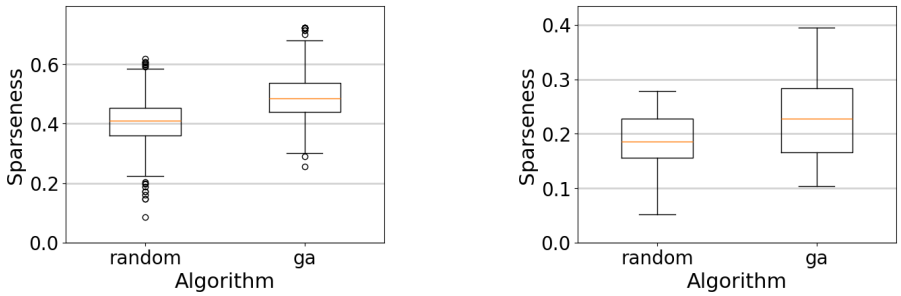
Figure 7.5 Number of failures revealed over time

Table 7.4 presents the sparseness metrics obtained by each search strategy, along with the results of the statistical significance tests comparing their differences. As can be seen, GA generates significantly more diverse test cases in terms of structural features compared to RS, with a large effect size. This can also be observed in Figure 7.6, which displays box plots of feature sparseness values for test cases generated by GA versus RS. Thus, GA not only increases the number of revealed failures by exploiting the characteristics of previous test cases but also enhances the diversity of features among these test cases. The randomness introduced through crossover and mutation in our implementation promotes novelty in new

generations, even though they are derived from previous test cases. In terms of severity sparseness, GA produces higher values than RS; however, there is no statistically significant difference. This indicates that both search strategies were effective in uncovering a variety of failure modes.

Table 7.4 Sparseness metrics obtained by each search strategy

Use case	Metric	Ran	GA	p-value	Effect size
uc1	Features Sparseness	0.406	0.488	< 0.001	-0.563 L
	Severity Sparseness	7.6	7.8	1	-0.12 N
uc2	Features Sparseness	0.188	0.235	0.023	-0.342 M
	Severity Sparseness	2.4	2.8	0.683	-0.28 S



(a) Use case 1 feature sparseness (b) Use case 2 feature sparseness

Figure 7.6 Sparseness of the structural features of the revealed failures for use cases 1 and 2

Expert feedback. The design of the robot tool and the pick-and-place movements significantly influence the vision system requirements. For the suction gripper use case, containing a big number of soft failures, the precise position and orientation detection play an important role in the task achievement, as most of the task failures occurred following the mispredictions of the DNN model. At the same time, the ARM with parallel gripper in UC-2 is more prone to hard failures. This is because the gripper, being relatively large (when opened), may collide with other objects during the picking or accidentally contact the other cardbox during movement, causing slight deviations from the intended placement position and augmenting the initial errors from the perception module. To compensate for the influence of external factors, the requirements for object detection precision by DNN should be increased.

RQ2: Usefulness of MARTENS in repairing DNN's inefficiencies

Motivation. We aim to assess the effectiveness of our approach in revealing failures that can be exploited to address inefficiencies in the DNN. We also seek to examine the persistent failures following the DNN repair to understand its limitations and identify potential design flaws.

Method. Based on the obtained failure and near-failures sets S_F and S_{NF}, the DL models are fine-tuned for each use case as described in Section 7.2.5. To compare the usefulness of our GA implementation with simple RS, we create separate fine-tuned models, M_f^{RS} and M_f^{GA}, which are trained exclusively on datasets collected during the execution of tests generated with their respective search strategies, RS and GA. Then, we re-run the test cases in the simulation using M_f^{RS} and M_f^{GA} for each use case. We calculate the number of passing, failing, and near-failing tests, noting the type of failures: soft or hard. We refer to the failed tests as 'non-repaired' tests, indicating that the failure persists even with the fine-tuned DNN. To account for the non-determinism of the simulator and ensure these failures are not due to simulator flakiness, we re-run each non-repaired test five times.

Results. Tables 7.5 and 7.6 present the statistics on the evaluation of the models fine-tuned with datasets generated by either GA or RS for both use cases.

Table 7.5 UC-1 test analysis after model repair

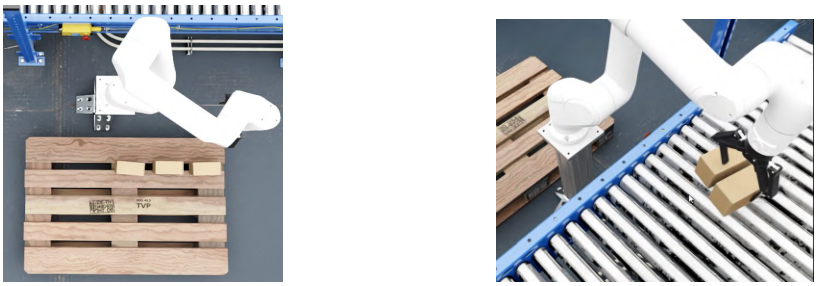
Algorithm	**All Tests**			**Failure Type**		**Total**
	Pass	**Fail**	**Near Fail**	**Soft**	**Hard**	
RS	557	1	0	0	1	558
GA	838	1	0	0	1	839
Total	1395	2	0	0	1	1397

Table 7.6 UC-2 test analysis after model repair

Algorithm	**All Tests**			**Failure Type**		**Total**
	Pass	**Fail**	**Near Fail**	**Soft**	**Hard**	
RS	59	0	0	0	0	59
GA	145	2	0	0	2	147
Total	204	2	0	0	2	206

For both use cases, most of the failures, more than 99% were successfully repaired, showing that the fine-tuning of the model improves the vision accuracy, so the soft failures are fixed. As expected, the non-repaired tests result from hard failures, indicating instances where

the robot failed to complete the task despite accurate cardbox location predictions by the vision model. Interestingly, we observed a reduction in the number of hard failures after repairing the vision model. This improvement underscores the relationship between enhanced precision in pose estimation and increased system robustness, i.e., a more precise vision model can compensate for certain flaws in the robot control design, thereby contributing to improved task completion rates. This reduction of hard failures by vision model repair is aligned with the feedback of experts on the previous RQ results.Given the reduced number of non-repaired tests, we conducted a detailed manual analysis to identify two primary root causes for these hard failures. In both use cases, one recurring issue involved the robot arm shaking before placing the box on top of the pallet, resulting in a misaligned placement. An illustration of such a non-repaired test with a misplaced cardboard box is provided in Figure 7.7a, with a corresponding video demonstration accessible via the provided link `https://youtu.be/NzXK4LTLnjo`. In addition, for UC-2, the robot was unable to pick objects when the distance between cardboard boxes was too narrow for the parallel gripper to maneuver around the object, as can be seen in Figure 7.7b and its respective scene video at the link `https://youtu.be/b23gYYBU9cU`.



(a) Misplaced boxes after shaking. (b) Boxes are too close for the gripper.

Figure 7.7 Illustration of non-repaired tests

Expert feedback. The generation of failure-revealing test cases is crucial for identifying the design's blind spots and weaknesses. These experiments demonstrate that the suction gripper is generally more suitable than the parallel gripper. As more test cases are simulated, situations arise where the robot cannot perform the pick with the parallel gripper. However, if there is insufficient air pressure for the suction gripper to function properly, several techniques can be implemented to improve the usability of the parallel gripper. For instance, an accelerated conveyor speed can create distance between passing products, or an additional movement can push the product in a certain direction before picking it up if it is in collision with another. Regarding the shaking of the robot, the controller uses Inverse Kinematics (IK)

to move the Tool Center Point (TCP) within the Cartesian space. However, this approach is based on numerical optimization, which can be subject to singularity issues, causing the robot to halt its movement. In such situations, the robot controller should move the joints directly without altering the TCP position to escape the singularity and continue the movement. This issue manifests quickly in the simulator as shaking, which affects the placement accuracy. A possible solution to improve the placement accuracy in this case would be adding a small time delay prior to place allowing the end effector to stabilize.

RQ3: Sim-to-real transferability of the in-simulation optimized DNN

Motivation. Although our proposed in-simulation testing effectively identifies ARM design flaws and limitations, our primary objective in optimizing the DNN within the simulation is to facilitate its transfer to the real-world with reduced costs.

Method. We collect and label two limited datasets, consisting of approximately 100 images each, from the actual production environments that conform to the simulation for both use cases. First, we evaluate our original DNN trained with synthetic data, denoted M_o, and our optimal, fine-tuned DNN (i.e., obtained by fine-tuning on GA-generated dataset), denoted M_f, for UC-1 and UC-2, on the real-world data. We compute the mean Average Precision (mAP), and incorporate the F1-score to quantify the effects of false negatives (i.e., undetected cardboard boxes) and false positives (i.e., irrelevant regions mistakenly identified as cardboard boxes). We evaluate the models on all the real-world data points. Secondly, we divide the real dataset into training (60%), validation (20%), and testing (20%) subsets, then, we used either M_o or M_f as a pre-trained DNN to train the vision model for detecting cardboard boxes in the real-world environment, referring to the resulting models as M_{or} and M_{of}, respectively. As a baseline, we train a YOLOv8 model using its standard pre-trained weights exclusively on the real-world training dataset, denoted as M_r. We compare the mAP and F1-score of each model on the real world test dataset for each use case, as well as the number of epochs required to converge to the best-fitted model. We consider the model converged if during 5 epochs there is no reduction in loss value. We also report the improvement scores I_{or} and I_{fr}, which compare the relative improvement in percentage for mAP and F_1 scores for the models F_o and F_{or} as well as F_f and F_{fr}. Our intuition is that the performance of the models on the real data should improve as real-world data examples are added to the fine-tuning dataset.

Results. Figure 7.8 shows the images of the scene set up in the real world. Table 7.7 reports the mAPs and F1-scores obtained by the models, M_o and M_f, on the real-world datasets.

(a) Use case 1 installation (b) Use case 2 installation

Figure 7.8 Real-world setups for both use cases

For UC-1, both models perform well on the real datasets by having performance metrics higher than 0.9. The fine-tuned model outperforms its original version. As can be seen in Figure 7.9, M_o fails to detect some cardboxes, or detects two colliding ones as the same. For UC-2, we found similar results, where M_f outperforms M_o, but the overall performance results are substantially lower than the UC-1. As shown in Figure 7.9, M_o fails to detect some cardboxes (i.e., false negatives) while M_f misidentifies irrelevant regions as cardboxes (i.e., false positives). Both models struggle to accurately delineate the boundaries of the cardboxes, resulting in a lower mAP compared to UC-1. Despite not being trained on real datasets, these models perform relatively well on real images.

Table 7.7 Performance of the model trained with synthetic data on the real world images

	Use case 1		**Use case 2**	
	M_o	M_f	M_o	M_f
mAP	0.911	0.915	0.723	0.822
F1	0.962	0.972	0.892	0.93

Table 7.8 shows the mAPs, F1-scores, and the number of epochs until convergence for models, M_r, M_{or} and M_{of}. Despite leveraging an established model like YOLOv8 and its built-in features, such as on-the-fly data augmentation, training an acceptable model with a limited real-world dataset proved challenging. This is indicated in the performance metrics obtained for both use cases in Table 7.8. As introduced in our research work, to achieve better results, a higher investment is necessary to establish an expert-guided protocol for data collection and labelling, ensuring larger real-world datasets with higher coverage for both normal and extreme operating conditions. Notably, for both use cases the fine-tuning on the real world data improved the mAP score compared to training on only synthetic data as seen from the improvement scores I_{or} and I_{fr}. M_{fr} converged on the real-world fine-tuning dataset of UC-1 after only six epochs of training, whereas M_{or} failed to achieve similar results even with more

(a) Use case 1 predictions with M_o

(b) Use case 1 predictions with M_f

(c) Use case 2 predictions with M_o

(d) Use case 2 predictions with M_f

Figure 7.9 Predictions on the real world data with the models trained on synthetic data for use case 1 and use case 2

epochs. Regarding UC-2, M_{fr} outperforms M_{or} in detection performance and M_{or} shows faster convergence. However, further improvements are necessary to develop a production-ready model for UC-2, as M_{fr} could not surpass a 0.9 mAP. Discussions with experts yielded the following feedback.

Table 7.8 Performance of the model trained on real data without and with pre-trained model on synthetic data

	Use case 1					**Use case 2**				
	M_r	M_{or}		M_{fr}		M_r	M_{or}		M_{fr}	
		M_{or}	$I_{or}(\%)$	M_{fr}	$I_{fr}(\%)$		M_{or}	$I_{or}(\%)$	M_{fr}	$I_{fr}(\%)$
mAP	0.806	0.936	2.7	0.948	3.6	0.755	0.871	20.4	0.892	8.51
F1	0.91	0.968	0.62	0.998	2.67	0.905	0.924	3.58	0.958	3
Epochs	26	9	-	6	-	26	9	-	12	-

Expert feedback. The optimized models in simulation successfully learned relevant in-

ductive biases quickly and efficiently in the production environment. These models were already familiar with the patterns of the cardboxes, the conveyor, and even the colors from the photorealistic simulator (Isaac Sim), as evidenced by their initial performance results on the collected real datasets. Fine-tuning was necessary because of the domain transfer from the simulation to the real world. The models were able to capture the variances of the real domain rapidly, in fewer than 10 epochs, compared to 26 epochs, when no pre-trained model was used. The challenge in solving the real-world version of UC-2 stems from the fisheye effect of the camera. The cardboxes in UC-2 have less width but greater height, creating a fisheye effect that negatively impacts pose estimation by distorting the detection of the box contours. Experts recommend either adjusting the virtual camera parameters in Isaac Sim to simulate a more natural fisheye effect, or modifying the real camera parameters and setup to reduce the fisheye effect on the cardboxes.

7.4 Chapter Summary

MARTENS is effective in revealing failures for ARM, allowing the detection of 26% to 50% more failures with higher diversity compared to random test generation. Over 99% of the revealed failures were fixed after fine-tuning the DNN model with data collected from these failures. The remaining failures were attributed to controller misconfiguration or environmental interactions. Following the analysis of these non-repaired tests, engineers proposed three possible design improvements. Using the model repaired with the MARTENS (fine-tuned on failures and real world data) as a pretrained base to perform inference in the real-world environments, we were able to achieve mAPs of 0.948 and 0.892 on test datasets for Use Case 1 and Use Case 2, respectively, after fewer than 10 epochs of fine-tuning. This compares to mAP values of 0.915 and 0.822 when only synthetic data was used, confirming the importance of accounting for real-world operational context.

CHAPTER 8 STRIM: SIM2REAL-AWARE IN-SIMULATION TESTING AND REPAIR OF MANIPULATOR VISION MODELS

8.1 Problem Context

Autonomous robotic manipulators (ARMs) increasingly rely on machine-learning-based vision to operate safely and efficiently in industrial cells. Cameras, mounted in the cell or on the robot, feed 2D images to deep neural networks (DNNs) for object detection and pose estimation, driving pick-and-place, palletization, and machine-tending [261]. In deployment, these models are expected to remain reliable across a variety of *foreseeable* conditions, including rare combinations of target object poses, lighting, clutter, and wear. Such long-tail situations may be infrequent but consequential: misdetections or mislocalizations can cause failed grasps, misplacements, or safety hazards.

Achieving long-tail robustness is a major testing challenge. Standard DNN-based perception QA includes collection of real images from the cell, manual annotation (e.g., polygons and angles), and evaluation with metrics such as F1 and mean Average Precision (mAP) [262]. Scaling this to low-frequency, safety-critical conditions is often impractical. Capturing edge cases is costly and disruptive, and high-quality annotation at scale is expensive. As a result, ARM vision models are often validated on test sets that under-represent rare but realistic conditions, leaving residual risk.

Photorealistic simulators, such as NVIDIA Isaac Sim, offer a compelling alternative. They provide automatic 2D/3D labels, parameterized control over scene factors (pose, materials, lighting), and large-scale data generation for training and testing. Prior work leverages synthetic inputs via transformations [241, 242], generative models [145, 243], and simulation-based testing with randomization, search, or reinforcement learning [34, 36, 97]. However, purely simulation-based pipelines remain limited by the simulation-to-reality (sim2real) gap: differences in texture, illumination, background clutter, and sensor characteristics can cause models that perform well in simulation to degrade on real camera streams unless adapted with production data. Many approaches implicitly assume this gap is small and report failures in simulation only [38, 39], without showing that simulation-driven improvements transfer to deployment.

We address these limitations by making simulation-driven training and testing more robust to domain drift. We target ARM perception models trained and stress-tested in a photorealistic simulator but deployed in a real industrial cell. The key idea is to insert an *unsupervised*

sim-to-real style-transfer model between the simulator and the vision model. The translator is trained on unpaired data: a small set of unlabeled real images from the target cell and a randomly generated set of simulator images. Once trained, it maps synthetic renderings into real-style images that better match the camera stream statistics while preserving scene geometry.

Building on this idea, we propose STRIM, Sim2Real-Aware In-Simulation Testing and Repair of Manipulator Vision Models, which couples style transfer, in-simulation training, search-based online testing, and data-driven repair. First, STRIM translates labeled synthetic images into real-style training data, allowing the detector to learn from abundant simulator labels in a deployment-aligned input domain. Second, STRIM performs closed-loop testing in simulation: a genetic algorithm (GA) searches scene parameters (object pose, lighting) to generate challenging cases, which are rendered, style-transferred, and evaluated within a simulated pick-and-place loop. Third, STRIM logs failure-inducing cases with sim-provided labels and fine-tunes the model, iterating test-and-repair to strengthen performance on realistic corner cases.

Unlike most simulation-based testing approaches, STRIM explicitly evaluates how in-simulation improvements transfer to real-world. This chapter makes the following contributions:

- We introduce STRIM, a sim2real-aware framework for in-simulation testing and repair of manipulator vision models that reduces the sim-to-real gap without relying on manually labeled real-world data.
- We integrate unpaired style transfer into simulation pipelines and compare CycleGAN, CUT, and FastCUT models to select an effective translator for robotic scenes.
- We propose a search-based testing and repair procedure over a parameterized scene space and compare GA against random sampling under a fixed budget for corner-case discovery.
- We evaluate STRIM on two industrial pick-and-place use cases in NVIDIA Isaac Sim as well as real world, showing that unpaired style transfer and search-based testing reduce the sim2real gap, uncover more failures than unguided exploration, and improve robustness using only unlabeled real images for style-transfer training.
- We release the collected datasets and trained models as open-source artifacts, which can serve as benchmarks for future research on sim-to-real gap minimization.

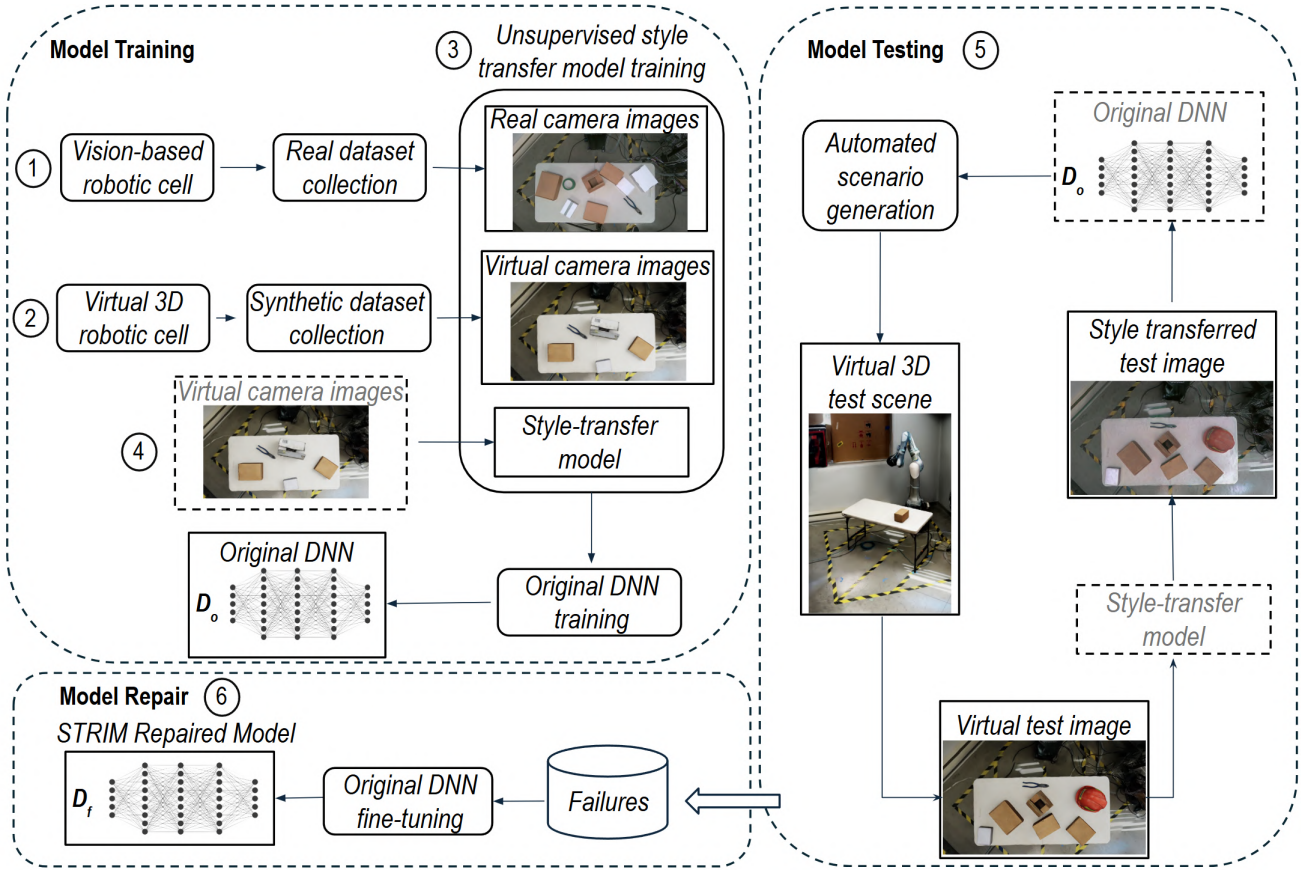


Figure 8.1 The diagram of the STRIM approach

8.2 Approach

To narrow the simulation-to-reality (sim2real) gap during in-simulation testing and improvement, we propose STRIM, Sim2Real-Aware In-Simulation Testing and Repair of Manipulator Vision Models. STRIM trains an *unpaired* sim-to-real style-transfer *translator* using unlabeled real images from the target cell and synthetic renderings from the simulator. The translator maps simulator images into real-style images that better match the deployment camera stream, enabling test-and-repair in simulation without relying on manually labeled real-world data. STRIM then couples this translator with GA-based search to generate corner cases in a parameterized scene space and to drive failure-driven repair.

Figure 8.1 summarizes the workflow. In Step 1, we collect unlabeled real-world images from the target robotic setup. In Step 2, we construct a virtual environment that corresponds to the target real-world scene in a 3D simulator. We use NVIDIA Isaac Sim [263] for its photorealistic rendering and support for dynamic simulation. This step includes creating the 3D assets present in the cell. While Isaac Sim provides libraries of common objects (e.g.,

boxes, tables, warehouse layouts), matching task-specific real assets with predefined models can be difficult. We therefore use 3D scanning to reproduce the real setup in simulation. We then collect a set of randomized images from this virtual environment. In Step 3, together with real-world images, simulated images are used to train an unpaired style-transfer model that serves as the sim-to-real translator. Once trained, the translator converts simulator renderings into real-style images that better align with the appearance distribution of the collected real data.

In Step 4, we train an initial object-detection DNN M_o on labeled synthetic images after their translation into the real-style domain. We then evaluate M_o in simulation (Step 5) under systematic scene variations generated by a genetic algorithm (GA). The GA proposes scene configurations (e.g., object pose and arrangement) within a parameterized search space. Each candidate is rendered by the simulator, translated into a real-style image by the sim-to-real translator, and then processed by M_o. The resulting predictions are used as feedback to guide the GA toward fault-revealing regions of the parameter space.

Finally, in Step 6, we use the simulator-provided labels for failure-inducing cases to fine-tune M_o, thereby improving robustness to realistic long-tail conditions. Overall, STRIM implements an iterative test-and-repair loop in which simulation, unpaired style transfer, and GA-guided search jointly identify and repair realistic failure cases, while reducing reliance on manually labeled real-world datasets.

In the following, we detail each component of the approach.

8.2.1 Steps 1–2: Real and Simulated Test Environments Setup

In the real-world environment setup phase, we collect a set of unlabeled real images D^{real} by operating the target robotic cell under nominal conditions and recording the RGB camera stream. Data collection is performed manually by varying the absolute and relative poses of the objects considered in this study.

In step 2, we create a photorealistic, parameterized simulator environment that approximates the target cell and exposes the variability needed to stress the vision system. We model the scene as a set of static objects $O_s = \{O_{s1}, \ldots, O_{sn}\}$ with fixed poses, and a set of flexible objects $O_f = \{O_{f1}, \ldots, O_{fk}\}$ whose poses vary across training and testing. The scene also includes a robotic arm R, a set of light sources $\mathcal{L} = \{L_1, \ldots, L_m\}$, and a camera C with a predefined field of view. The static set O_s defines the cell layout, while O_f contains the targets whose configuration is randomized to induce realistic long-tail conditions.

In principle, O_s and O_f can be instantiated from computer-aided design (CAD) models.

In practice, manual asset creation is labor intensive and requires specialized 3D modeling expertise [264]. Moreover, existing asset libraries, including those provided by Isaac Sim, often fail to match the geometry and appearance of the specific target cell and its objects. To reduce this mismatch, we use 3D scanning to build accurate virtual representations. Recent tools can reconstruct geometry from smartphone depth sensors (e.g., LiDAR) with sufficient fidelity for our use case [**?**]. We first scan the cell environment to obtain assets for O_s, then scan each flexible object in O_f individually. Since our focus is perception testing rather than motion planning or control, we do not actuate the robot during these experiments. We therefore treat the arm R as part of the static set O_s and include it as a scanned asset. Beyond the flexible targets, we include auxiliary distractor objects $O_d = \{O_{d1}, \ldots, O_{dl}\}$ that are not part of the classification set, but introduce realistic background clutter. Their poses are randomized alongside O_f, and they are also scanned individually. We detail these objects in Section 8.3.1. To approximate real lighting, we instantiate light sources in Isaac Sim using its available primitives [246] (e.g., rectangular, distant, spherical, cylindrical) and place them to mirror the physical cell. For example, if the room contains four ceiling-mounted lamps, we place four rectangular lights at the corresponding locations in the simulated environment. During testing, we randomize lighting parameters to vary illumination.

The camera C produces RGB renderings that serve as inputs to the DNN-based perception model. The camera can be fixed in the scene (*eye-to-hand*) or attached to the robot (*eye-in-hand*). In our study, we use an eye-to-hand camera mounted above the manipulable objects. Isaac Sim can also provide per-object ground truth (e.g., class and pose) for labeled synthetic data generation and for annotating failure-inducing cases during repair. Note that labels are not required when training the unpaired sim-to-real translator. We match the camera parameters, such as focal length and field of view, in the simulator to those of the real-world camera and adjust them accordingly to obtain a similar view of the objects.

Ultimately, the robotic system is intended to perform pick-and-place operations on the flexible objects O_f. Successful operation depends on reliable perception under long-tail conditions, where misdetections or pose estimation errors can lead to grasp failures. Our environment specification and parameterization therefore target the key sources of deployment variability, including illumination changes and pose variations of objects in O_f and O_d.

After the virtual environment setup, we use it to render a set of synthetic images D^{syn}, sampling scene variations through domain randomization over object pose and lighting. At this stage, we ignore simulation-provided labels and use only RGB renderings.

8.2.2 Step 3: Unpaired Sim-to-Real Style Transfer

In this step, we learn an image-level sim-to-real mapping that reduces the visual discrepancy between synthetic and real images without manual labeling. We use an unpaired, unsupervised image-to-image translation model that performs style transfer between the synthetic and real domains using unlabeled real images and synthetic renderings.

We train an unsupervised, unpaired style-transfer model on $(D^{\text{syn}}, D^{\text{real}})$ to learn a sim-to-real *translator*. The goal is to fit a mapping $G_{\text{S}\to\text{R}}$ that converts synthetic renderings into *real-style* images whose appearance matches the distribution of D^{real}, and optionally a reverse mapping $G_{\text{R}\to\text{S}}$. Since training is fully unpaired, it requires neither image-level correspondences nor task labels. After training, $G_{\text{S}\to\text{R}}$ is applied to simulator renderings to produce deployment-aligned, realism-enhanced inputs for subsequent training and GA-based testing.

This phase is performed once per target cell configuration and reused across STRIM's training and testing stages.

8.2.3 Step 4: Supervised Training on Style-Transferred Synthetic Data

This phase trains an object detector whose inputs are aligned with the real image domain while leveraging the precise, low-cost labels available in simulation.

Synthetic data generation and automatic labeling

We use NVIDIA Isaac Sim as the simulation platform, which provides GPU-accelerated physics simulation and photorealistic rendering. The simulator exposes environmental parameters and produces per-frame metadata, including depth maps and 2D annotations via color-based segmentation. For our 2D pick-and-place setting, we post-process the segmentation outputs to extract contours for the target objects O_f. We then represent each instance as a polygon and use its vertex coordinates as ground-truth annotations paired with each RGB image.

Using our customized scene-parameterization pipeline, we generate labeled synthetic images by randomizing environmental parameters within predefined distributions. This yields scenes with diverse object positions and orientations (x_i, y_i, r_i) as well as varying illumination levels l. The resulting dataset covers a broad range of configurations and supports supervised training of an object detection model.

Training on style-adapted synthetic data

Rather than training directly on raw simulator renderings, we first translate each synthetic RGB image into the real-style domain using the learned sim-to-real translator $G_{\mathrm{S}\to\mathrm{R}}$. Given a labeled synthetic sample (I^{syn}, y), where y encodes object classes and polygon coordinates, we construct a style-adapted sample $(\tilde{I}^{\mathrm{real}}, y)$ with $\tilde{I}^{\mathrm{real}} = G_{\mathrm{S}\to\mathrm{R}}(I^{\mathrm{syn}})$. The annotations y remain valid because translation changes image appearance while preserving the underlying scene geometry.

We train the supervised object detection model M on the style-adapted dataset $\tilde{D}^{\mathrm{train}} = \{(\tilde{I}^{\mathrm{real}}, y)\}$. We follow standard training and validation practice for object detection and record mean Average Precision (mAP) on a synthetic validation set processed through the same translator. This yields a detector trained on deployment-aligned, real-style inputs while retaining the labeling efficiency and simulation-provided coverage.

8.2.4 Step 5: Search-based Test Generation and Evaluation

This phase goes beyond unguided domain randomization by exploring the space of environmental conditions to uncover fault-revealing test cases. We perform search-based test generation using a genetic algorithm (GA), a standard choice in prior work on search-based testing [38,247]. Concretely, we implement the GA with `Pymoo` [179] and use AmbieGen [185] tool to specify the search problem and its parameterized scene space.

Search space and individual representation

In-simulation test generation requires programmatic control over scene factors that affect vision-driven behavior. Given the environment in Section 8.2.1, we parameterize (i) the pose and type of flexible targets, (ii) the pose and type of auxiliary distractors, and (iii) lighting intensity.

The scene contains N_f flexible objects drawn from O_f and N_d auxiliary (distractor) objects drawn from O_d, along with N_L light sources. In our study, each object type corresponds to a distinct appearance (e.g., shape or color), and the scene contains at most one instance per type. We encode each candidate test (individual) as a one-dimensional chromosome of length $4(N_f + N_d) + N_L$. The first $4(N_f + N_d)$ genes represent object parameters as tuples $[x_i, y_i, r_i, t_i]$. Here, (x_i, y_i) denotes the object position in the workspace plane, with the z-coordinate fixed to a predefined value; r_i is the rotation about the z-axis; rotations about the x- and y-axes are fixed to 0; and t_i encodes the object type, selected from $[0, (N_f + N_d))$. The remaining N_L genes encode the luminosity levels l_m for the light sources L_m. All genes

are normalized to $[0, 1]$.

We enforce feasibility constraints to bound the search space and avoid invalid scenes: (i) objects do not intersect, (ii) object positions remain within the pick-and-place region, and (iii) at least one target from O_f is present. Candidates that violate any constraint are rejected and are not evaluated by the search.

Fitness function

The fitness function quantifies how well an individual supports the search objective and drives selection across generations. A binary success or failure signal is too sparse to effectively guide the GA. We therefore use a continuous signal derived from the perception errors of the DNN under test M.

Given an individual $\mathbf{x}$, we instantiate the corresponding scene configuration in Isaac Sim. The simulator renders an RGB image I^{syn} and outputs ground-truth semantic segmentation s'. We translate the rendering into the real-style domain via $\tilde{I}^{\text{real}} = G_{\text{S}\to\text{R}}(I^{\text{syn}})$, and then apply the detector M to obtain predicted bounding boxes $\{\hat{b}_n\}_{n\in\mathcal{N}}$. We compute the ground-truth bounding boxes $\{b_i\}$ by converting the segmentation output into per-object boxes. The fitness is defined as:

$$f(\mathbf{x}) \;=\; \min_{i\in\{1,\ldots,N_f\}} \text{IoU}\Big(\hat{b}_i(\mathbf{x}),\, b_i\Big) \tag{8.1}$$

In Equation (8.1), b_i denotes the ground-truth box of the i-th target object and $\hat{b}_i(\mathbf{x})$ denotes the corresponding prediction under the scenario encoded by $\mathbf{x}$. We use the minimum Intersection-over-Union (IoU) [265] across target objects so that the fitness is dominated by the worst-localized instance. Lower values of $f(\mathbf{x})$ therefore correspond to more challenging scenarios that induce larger detection errors. The GA minimizes $f(\mathbf{x})$, seeking scenes where predicted and ground-truth boxes have minimal overlap.

Genetic operators and diversity preservation

We use one-point crossover, a standard operator in genetic algorithms [209]. Given two parent chromosomes, we sample a crossover point uniformly at random and exchange the gene suffixes to produce two offspring. After crossover, we check feasibility and keep an offspring only if it satisfies all validity constraints. Otherwise, we retain the corresponding parent.

We employ three mutation operators: replacement m_r, addition m_a, and removal m_d. The

replacement operator m_r selects an object tuple (x_i, y_i, r_i, t_i) and resamples its attributes. The addition operator m_a introduces a new object with randomly sampled attributes, while the removal operator m_d removes an existing object, provided that more than one object is present. At each generation, we apply mutation with probability p_{mut}. After mutation, we enforce validity constraints and admit only feasible individuals into the population.

To promote diversity among individuals, we remove near-duplicates at each iteration. We quantify the deviation between two individuals $\mathbf{x}$ and $\mathbf{y}$ using the Euclidean distance $D_e(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$ and compare it against a duplicate threshold D_{th}. Since decision variables are normalized to $[0, 1]$ and the chromosome length d may vary, we use the normalized distance $D_{\text{norm}}(\mathbf{x}, \mathbf{y}) = D_e(\mathbf{x}, \mathbf{y})/\sqrt{d}$ and discard one of the individuals if $D_{\text{norm}}(\mathbf{x}, \mathbf{y}) < \varepsilon$. Here d corresponds to the size of the bigger chromosome in the compared pair.

Test execution and failure classification

For each evaluated individual (test) $\mathbf{x}$, the instantiated scene produces: (i) a real-style image $\tilde{I}^{\text{real}}$ obtained via sim-to-real translation, (ii) the predictions of the model under test M, and (iii) simulator-provided ground-truth annotations. We classify a test case as a failure if the IoU of at least one target object falls below 0.95, and we record all such failed test cases in the set S_F.

As a baseline, we also implement Random Search (RS), which generates test cases by sampling chromosome values uniformly within their feasible ranges. Under the same budget, we expect GA-based search to uncover more failures than RS while still maintaining diversity among the discovered failures.

8.2.5 Step 6: Failure-Driven Model Repair

The final phase of STRIM repairs the vision model using the failure cases discovered during GA-based testing. The repair set comprises all real-style images $\tilde{I}^{\text{real}}$ associated with failed tests, together with Isaac Sim-provided annotations.

For each failed test case in S_F, we store the translated image $\tilde{I}^{\text{real}}$ and derive labels from the simulator segmentation output. After post-processing the segmentation masks, we obtain a labeled failure dataset D_F suitable for supervised fine-tuning.

Rather than retraining from scratch, we fine-tune the current model M on D_F using transfer learning. We reuse the initial training hyperparameters, but reduce the maximum number of epochs to avoid overfitting to failure samples and to mitigate catastrophic forgetting.

By iterating search-based test generation (Step 5) and failure-driven repair (Step 6), STRIM progressively hardens the perception model against realistic corner cases discovered in simulation, while avoiding reliance on manually labeled real-world data.

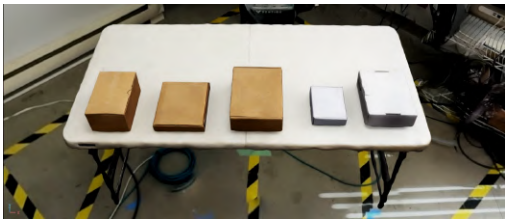
8.3 Evaluation

8.3.1 Experimental Setup

In this subsection we detail the experimental setup, including the use cases, training hyperparameters, algorithm configurations, and evaluation metrics.

Industrial Use Cases

We evaluate STRIM on two industrially motivated pick-and-place scenarios (UC1, UC2) that rely on object detection in robotic cells. In both use cases, the perception task is to detect *oriented* bounding boxes that tightly enclose cardboard boxes belonging to a set of supported classes. Each oriented box is parameterized by the 2D center coordinates, an in-plane rotation angle (yaw about the camera z-axis), and its width and height. Examples of the target boxes are shown in Figure 8.2a (UC1) and Figure 8.2b (UC2). For reporting and analysis, we index the target boxes from left to right as $box_1, \ldots, box_N$ (e.g., box_5 in UC1 corresponds to the rightmost white box in Figure 8.2a). UC2 involves larger boxes representative of palletization settings, where the robot transfers boxes from a surface onto a pallet. Due to confidentiality constraints, we cannot release the real industrial 3D environment and imagery. We therefore reproduce the setup in a controlled development environment using a representative table-top scene while preserving key characteristics of the real cell. To increase realism, we incorporate domain knowledge from robotics engineers: in practice, non-target objects may be present near the boxes and some boxes may be open. We include seven distractor objects, shown in Figure 8.2c (two tapes, two protective helmets, pliers, and two open cardboard boxes), randomly placed alongside the target boxes. These distractors must be ignored by the perception module, as false detections can lead to unsafe or incorrect manipulation actions. The model under test is therefore evaluated not only on its ability to localize target boxes accurately, but also on its ability to avoid reporting irrelevant objects as pick candidates.



(a) Cardboxes in UC1

(b) Cardboxes in UC2

(c) Additional objects

Figure 8.2 Objects used in both use case to test the model

Generative model training

We consider two families of unpaired image-to-image translation models, CycleGAN [59] and CUT (including its FastCUT variant) [65]. We use the public implementations provided in the authors' repository [266] and follow the recommended hyperparameters. Specifically, we train with Adam optimizer [222] using an initial learning rate of 2×10^{-4} and a linear learning-rate decay schedule. The generator follows a ResNet-based architecture [267], and the discriminator uses a PatchGAN design [268]. To control GPU memory usage, input images are resized while preserving aspect ratio to a width of 960 pixels and randomly cropped to 360×360 during training.

Perception model training

We use YOLOv11 [269] as the object detector, leveraging its support for oriented bounding boxes. Training annotations are provided as normalized polygon coordinates tightly enclosing each target object. These polygons are extracted from the instance-segmentation buffers produced by Isaac Sim and converted to oriented-box labels. In UC1, the model is trained as a five-class detection DNN, whereas in UC2 it is trained to detect three classes, corresponding to the defined cardboard box types. For real-world data collection, we use an Intel RealSense D435 camera [252]. For consistency, we also employ the corresponding simulated model of the same camera for data collection in simulation.

We follow the training configuration recommended by the YOLOv11 authors: batch size 16 and an adaptive learning rate in the range $[10^{-2}, 10^{-4}]$. Input images are resized while preserving aspect ratio from 1280×720 to a width of 960 pixels. Models are trained for up to 100 epochs with early stopping (patience = 30) based on validation performance. To reduce confounding factors in the evaluation, we disable all detector-side image augmentations during training.

Search algorithm configuration

For both use cases, we adopt a variable-length chromosome that encodes the scene description vector. Each scene may contain up to N_{f} target cardboard boxes and $N_{\mathrm{d}} = 7$ distractor objects. In UC1, $N_{\mathrm{f}} = 5$, yielding a maximum chromosome length of $(N_{\mathrm{f}} + N_{\mathrm{d}}) \times 4 + 2 = 50$ genes and a minimum length of 6. In UC2, $N_{\mathrm{f}} = 3$, resulting in a maximum length of 42 and the same minimum of 6. Maximal-length chromosomes correspond to scenes where all objects are simultaneously present on the surface, which is uncommon because validity constraints prevent object intersections. Each object contributes four genes encoding its planar pose and identity: position (x_i, y_i) in meters, in-plane rotation r_i in degrees, and object type t_i. Two additional genes encode the illumination levels of internal and external light sources, denoted l_1 and l_2 (relative units), corresponding to artificial and natural light in the room. The ranges of all genes are provided in Table 8.1.

Table 8.1 Allowed ranges for the parameters

x_i	y_i	r_i	t_i	l_1	l_2
$0 - 0.6$	$0 - 0.3$	$0 - 180$	0 - $(N_{obj}$ + $N_{box})$	$1 - 15$	$1 - 15$

We use a mutation probability $p_{\mathrm{mut}} = 0.4$, crossover rate $p_{\mathrm{cross}} = 0.9$, and de-duplication threshold $D_{th} = 0.02$, as recommended in former search-based testing work [157] and confirmed via warm-up experiments. We set the population size to 100 and offspring of 50 per generation. Each run is executed under a fixed evaluation budget of 1000 test scenes.

Metrics

Style-transfer quality. To evaluate the style-transfer generative models, we use the kernel inception distance (KID) metric [270], which is better suited for smaller datasets than common alternatives such as Fréchet Inception Distance (FID) [271] and Inception Score (IS) [272]. KID is computed by passing both real and generated images through a pretrained

convolutional neural network, typically Inception-v3 [273], to extract high-level semantic features (commonly 2048-dimensional). KID is then computed as the squared Maximum Mean Discrepancy (MMD) [274] between the feature distributions of real and generated images. MMD is a kernel-based statistical distance between two distributions that does not assume Gaussianity of the features, unlike FID, making KID more reliable under limited data. Lower KID indicates a closer match to the target image distribution.

Detection accuracy. For object detection, we report the COCO-style mean Average Precision, denoted $\mathrm{AP}_{50\text{–}95}$. Let $\mathrm{IoU}(b, \hat{b})$ be the intersection-over-union between a ground-truth box b and a predicted box $\hat{b}$. For an IoU threshold t, a prediction is a true positive if it matches an unmatched ground-truth box with IoU $\geq t$ (using standard one-to-one matching), otherwise it is a false positive; any unmatched ground-truth box is a false negative. For each $t \in \{0.50, 0.55, \ldots, 0.95\}$, we compute AP_t as the area under the precision–recall curve obtained by sweeping the detector confidence threshold. We then average across thresholds:

$$\mathrm{AP}_{50\text{–}95} \;=\; \frac{1}{10} \sum_{t \in \{0.50, 0.55, \ldots, 0.95\}} \mathrm{AP}_t.$$

This metric jointly reflects detection quality across confidence levels and localization strictness.

Detection balance. We also report the F1 score to summarize the precision–recall trade-off. For a fixed IoU threshold τ and the same one-to-one matching protocol, let TP, FP, and FN denote the numbers of true positives, false positives, and false negatives, respectively. Here precision, recall and F1 score are

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \quad R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \quad \mathrm{F1} = \frac{2PR}{P + R}.$$

In our experiments, we compute F1 per target box category and report the average across categories.

Testing model effectiveness and robustness. We report the number of *distinct* failures discovered under a fixed search budget. To avoid counting near-duplicates, we apply Euclidean distance-based de-duplication on normalized scene vectors and count a failure as unique only if its distance to all previously recorded failures exceeds $D_{th} = 0.02$.

Statistical analysis and significance testing. When comparing stochastic search algorithms, we repeat each configuration at least five times. We report pairwise, non-parametric Mann–Whitney U tests with significance level $\alpha = 0.05$, together with effect sizes measured using Cliff's delta [182, 183]. Statistical results are visualized in the boxplots: horizontal brackets indicate pairwise comparisons, stars denote significance ("**": $p < 0.01$, "*":

$p < 0.05$), and dots denote effect size magnitude (“..”: large, “.”: medium).

Hardware. We run experiments on an AWS `g5.2xlarge` VM [258] equipped with an NVIDIA A10G Tensor Core GPU (24 GB VRAM), 8 vCPUs (AMD EPYC), and 32 GB RAM.

8.3.2 Results

RQ1 – Selecting a sim2real transfer model

Motivation. Unsupervised style transfer is a core component of STRIM. This RQ compares candidate generators and selects the model that best translates simulated images into a real-like domain, thereby minimizing the distribution shift with respect to real camera data.

Method. We evaluate three unpaired image-to-image translation models, CycleGAN [59], CUT, and FastCUT [65], as well as a no-transfer baseline (Sim). For each use case (UC1, UC2), we collect 200 real images $\mathcal{D}_{\text{real}}$ from the robotic cell and expand them to 1000 samples using label-preserving augmentations: random rotations within $\pm 10^{\circ}$ and color jitter with up to 20% variation in brightness, contrast, or saturation. Following the dataset size balancing recommendation in [275], we also generate 1000 simulated images $\mathcal{X}_{\text{sim}}$ from the digital twin. Each style-transfer model is trained for 100 epochs.

To quantify alignment with the real domain, we generate an additional set of 200 simulated images $\mathcal{D}_{\text{sim}}$ and translate it with each trained model, yielding $\mathcal{D}_{\text{CG}}$, $\mathcal{D}_{\text{CUT}}$, and $\mathcal{D}_{\text{FCUT}}$. We then measure the distributional gap between $\mathcal{D}_{\text{real}}$ and each transferred set using KID (lower is better). We repeat KID computation five times, each time sampling 100 images uniformly at random, and report mean and standard deviation in Table 8.2. Training time is reported in Table 8.3. Finally, we qualitatively inspect representative transfers (Figure 8.3 to Figure 8.5) to verify that geometry and label-relevant structure are preserved while realism improves.

Results. Table 8.2 shows that all three generators substantially reduce KID relative to Sim for both UC1 and UC2, indicating that style transfer narrows the sim2real gap. CycleGAN and CUT improve KID by more than 45% over the Sim baseline, while FastCUT achieves the lowest KID in both use cases, suggesting the closest match to the real-image style. Table 8.3 further shows that FastCUT is the most efficient, requiring roughly 50% less training time than the alternatives. This reduction is consistent with FastCUT removing the identity loss used in CUT, which otherwise adds forward passes on target-domain images [65]. Qualitative comparisons (Figure 8.3 to Figure 8.5) corroborate the quantitative results: FastCUT best matches real lighting and texture while preserving object boundaries and scene layout.

Table 8.2 KID scores (mean ± std) for two use cases.

Method	UC1	UC2
Sim	0.300 ± 0.015	0.3388 ± 0.0103
CycleGan	0.1621 ± 0.0044	0.2181 ± 0.0031
CUT	0.1680 ± 0.0021	0.2027 ± 0.0053
FastCUT	0.1584 ± 0.0057	0.1857 ± 0.0043

Table 8.3 Training time for the compared generative models.

Training time, min	UC1	UC2
CycleGan	428.75	417.55
CUT	435.81	424.2
FastCUT	225.8	220

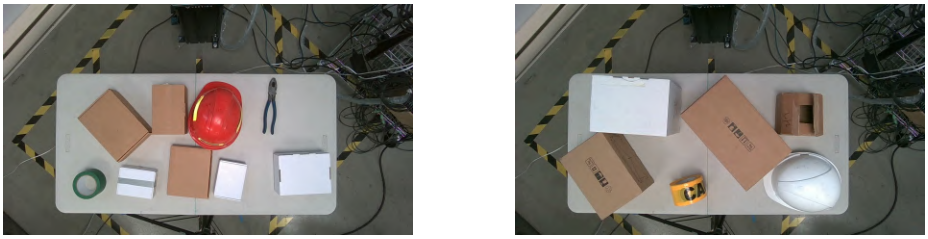
Conclusion. Based on KID and visual inspection across both use cases, we select FastCUT as the default style-transfer model for STRIM in subsequent experiments.

RQ2 – Utility of sim2real style transfer for real-world model deployment

Motivation. This RQ tests whether training on style-transferred synthetic images improves real-world detection performance, relative to training on raw simulated images captured with a virtual camera.

Method. For each use case (UC1, UC2), we train the same box-detection model under two data-generation pipelines. (*Sim*) The conventional pipeline uses 1200 randomly generated, annotated Isaac Sim images, split into 800/200/200 for train/val/test. (STRIM) We apply FastCUT to the synthetic images and train on the translated images while keeping the simulator-provided bounding-box annotations unchanged. We evaluate both trained models on the same held-out real test set of 200 images using $AP_{50–95}$ and the box-level F1 score, averaged across boxes.

Results. In UC1 (Table 8.4), Sim achieves $AP_{50–95} = 0.961$ and F1 = 0.846 on real images, whereas STRIM yields $AP_{50–95} = 0.962$ and F1 = 0.964. While $AP_{50–95}$ changes marginally, the F1 increase indicates a markedly better precision–recall balance on real data. In UC2 (Table 8.5), we observe the same pattern with stronger gains: STRIM improves $AP_{50–95}$ from 0.770 to 0.778 and increases F1 from 0.683 to 0.794. Overall, training on FastCUT-transferred images yields consistent improvements on real deployment data, with the main effect captured by F1 (gains of 14%–16%), indicating fewer missed detections under clutter/occlusion without



(a) Test subject 1 view (b) Test subject 2 view

Figure 8.3 Top views of the real world subjects under test with predictions

inducing spurious boxes.

Table 8.4 Performance of DNN model trained with simulated and FastCUT filtered images on real world data for UC1

Method	Metric	Box 1	Box 2	Box 3	Box 4	Box 5	Avg
Sim	AP50-95	0.957	0.885	**0.988**	**0.988**	0.988	0.961
	F1	0.836	0.697	0.787	**0.999**	0.911	0.846
FastCUT	AP50-95	**0.965**	**0.892**	0.981	0.983	**0.989**	**0.962**
	F1	**0.940**	**0.908**	**0.988**	0.991	**0.996**	**0.964**

Table 8.5 Performance of DNN model trained with simulated and FastCUT filtered images on real world data for UC2

Method	Metric	Box 1	Box 2	Box 3	Avg
Sim	AP50-95	0.913	0.671	0.726	0.77
	F1	0.569	0.697	0.784	0.683
FastCUT	AP50-95	**0.927**	**0.674**	**0.733**	**0.778**
	F1	**0.604**	**0.781**	**0.996**	**0.794**

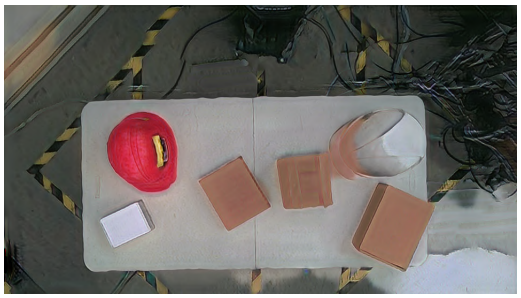
Conclusion. Training on FastCUT-transferred synthetic data reduces the sim2real performance gap for box detection on real images, motivating its use as the default in STRIM.

RQ3 – In-Simulation automated testing with evolutionary and random search

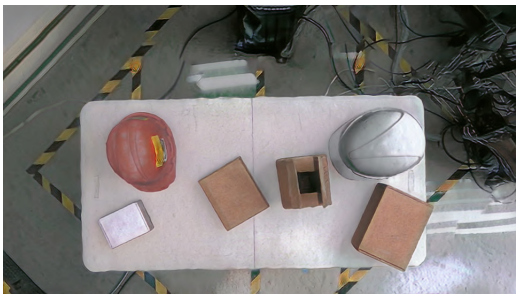
Motivation. While large-scale random generation can approximate the training distribution, effective testing must target rare yet plausible corner cases that are likely to expose model brittleness. This RQ evaluates whether evolutionary search in simulation yields more fault-revealing test scenes than random sampling for the detection model trained with STRIM.

(a) Simulated image

(b) CycleGAN transfer



(c) CUT transfer



(d) FastCUT transfer

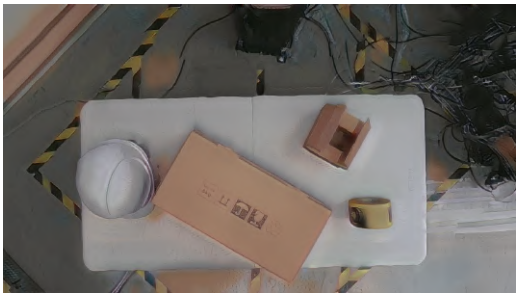
Figure 8.4 Comparison of different style transfer models

Method. We parameterize each simulated scene by a scene description vector that specifies object poses, object types, and lighting parameters, which Isaac Sim uses to instantiate a 3D scene and render an image. We search this space using random search and a genetic algorithm. For each algorithm, we perform 10 independent runs under the same evaluation budget of 1000 candidate scenes per run.

To assess fault-revealing capability, we deploy a perception module that uses the STRIM-trained detector M_o. Each candidate image rendered in simulation is translated to the real-domain style using FastCUT, matching the model's training distribution, and then passed to M_o. We record the number of distinct failed test cases per run. To avoid inflating failure counts, we apply a de-duplication mechanism that discards failures that are sufficiently similar to previously recorded ones, retaining only unique failures. We summarize the per-run failure counts using boxplots (Figure 8.6), including pairwise statistical comparisons and significance test results.

Results. Figure 8.6 shows that GA consistently discovers more unique failures than RS under the same budget in both use cases. In UC1, RS finds 71 failures on average, whereas GA finds 104, corresponding to a $\sim$47% increase. In UC2, GA increases the average number of failures from 343 to 400, an improvement of $\sim$16%. These results indicate that evolution-

(a) Simulated image



(b) CycleGAN transfer

(c) CUT transfer

(d) FastCUT transfer

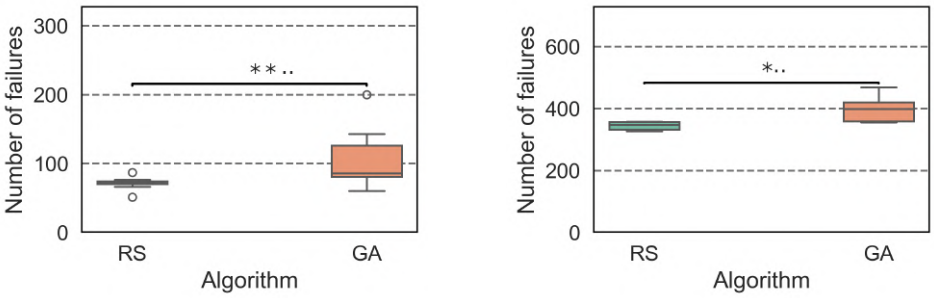
Figure 8.5 Comparison of different style transfer models

ary search more effectively concentrates evaluations on fault-revealing regions of the scene space, leveraging information from previously generated scenes to iteratively generate harder test cases. In contrast, RS more closely reflects average-case sampling and is less likely to repeatedly hit worst-case configurations within a fixed budget.

Conclusion. Evolutionary search is more effective than random sampling at uncovering unique failures for the STRIM-trained perception module. This motivates using GA as the default searching strategy in STRIM and, in the next RQ, evaluating whether its discovered failures can be used to repair the model and improve performance.

RQ4 – Model performance after fine-tuning

Motivation. This RQ evaluates whether the unique failures uncovered by GA-based simulation testing can be leveraged to *repair* the deployed detector. Concretely, we study whether fine-tuning the original model under test M_o on failure-inducing cases improves (i) robustness under the same in-simulation testing procedure and (ii) detection performance on real



(a) Number of failures for UC1 (b) Number of failures for UC2

Figure 8.6 Revealed failures using RS vs GA algorithm

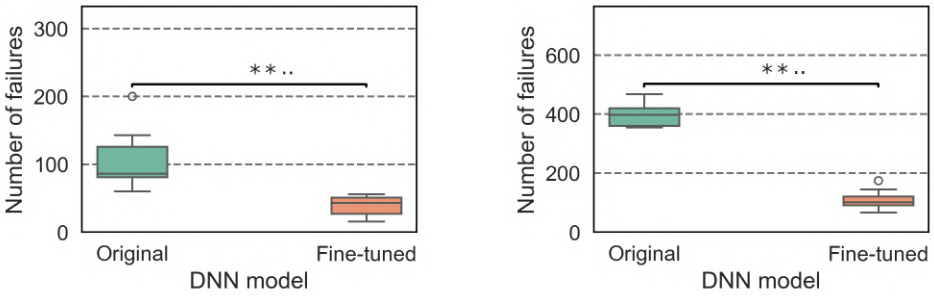
camera images.

Method. We aggregate all distinct failures recorded across the GA runs (RQ3) into a failure-focused dataset for each use case. We split this dataset into training and validation subsets and fine-tune the original detector M_o for up to 100 epochs with early stopping based on validation loss (patience = 30), yielding the repaired model M_f. Following the STRIM training pipeline, we translate the failure images using FastCUT before training, while keeping the simulator-provided bounding-box annotations unchanged.

We then evaluate M_f in two settings. (i) *In simulation:* we rerun the GA-based automated testing procedure (same budget and de-duplication as RQ3) and compare the number of unique failures revealed for M_f versus M_o. (ii) *In the real world:* we evaluate M_f on the held-out real test set (200 images) using $AP_{50–95}$ and box-level F1 (computed per box and averaged across boxes), and we report the same metrics for M_o and for the baseline model trained only on raw synthetic data.

Results. We fine-tune M_o using 1000 discovered failures for UC1 and approximately 3500 failures for UC2, producing M_f. Figure 8.7 compares GA-based failure counts before and after repair. In UC1 (Figure 8.7a), the average number of unique failures decreases from 104 to 38 per 1000 evaluated scenarios, a 2.7× reduction. In UC2 (Figure 8.7b), failures decrease from 400 to 107, a 3.7× reduction. These results show that fine-tuning on failure-inducing cases substantially improves robustness under the same adversarial testing procedure used to expose brittleness.

On real images, Table 8.6 shows that M_f achieves strong and consistent performance in UC1, with $AP_{50–95}$ between 0.928 and 0.995 (average 0.976) and F1 between 0.917 and 0.997 (average 0.968). Relative to M_o, this corresponds to a modest improvement of about 1.5% in average AP. In UC2 (Table 8.7), performance remains strong but more variable across



(a) Number of failures for UC1 (b) Number of failures for UC2

Figure 8.7 Revealed failures for original vs fine-tuned models

boxes, with $AP_{50–95}$ between 0.711 and 0.928 (average 0.797) and F1 between 0.523 and 1.000 (average 0.801), yielding an overall AP increase of 2.4% compared to M_o.

Table 8.6 Performance of CUT fine tuned across 5 boxes. Values rounded to three significant digits.

Method	**Metric**	**Box 1**	**Box 2**	**Box 3**	**Box 4**	**Box 5**	**Avg**
Sim	AP50-95	0.957	0.885	0.988	0.988	0.988	0.961
	F1	0.836	0.697	0.787	**0.999**	0.911	0.846
STRIM Trained Model	AP50-95	0.965	0.892	0.981	0.983	0.989	0.962
	F1	0.940	0.908	0.988	0.991	**0.996**	0.964
STRIM Fine-tuned Model	AP50-95	**0.974**	**0.928**	**0.995**	0.992	**0.989**	**0.976**
	F1	**0.948**	**0.917**	**0.992**	**0.997**	0.988	**0.968**

Method	**Metric**	**Box 1**	**Box 2**	**Box 3**	**Avg**
Sim	AP50-95	0.913	0.671	0.726	0.77
	F1	0.569	0.697	0.784	0.683
STRIM Trained Model	AP50-95	0.927	0.674	**0.733**	0.778
	F1	**0.604**	0.781	0.996	0.794
STRIM Fine-tuned Model	AP50-95	**0.928**	**0.752**	0.711	**0.797**
	F1	0.523	**0.879**	**1.000**	**0.801**

Table 8.7 Performance of CUT fine tuned across three boxes. Values rounded to three significant digits.

The larger gains observed in simulation than on real images are consistent with two factors. First, the real test set is limited in size and may not cover many of the corner cases targeted during GA-based testing and repair. Second, a residual sim2real gap persists, so some failure

modes optimized in simulation may not fully translate to the statistics of real imagery and real object appearance.

Conclusion. Fine-tuning on GA-discovered failures effectively repairs the detector, yielding robustness gains under automated simulation testing and consistent, albeit smaller, improvements on real deployment data. This supports the repair phase of STRIM as a practical mechanism to close the loop between simulation-based failure discovery and model improvement.

8.3.3 Threats to Validity

We discuss threats to validity and mitigation measures.

Internal validity. Our evaluation compares multiple style-transfer models (CycleGAN, CUT, FastCUT), training pipelines (raw vs style-transferred synthetic data), and testing strategies (RS vs GA). Outcomes can be influenced by implementation details and hyper-parameters, including generator architectures, crop size, learning-rate schedules, YOLOv11 training configuration, search operators, and the failure de-duplication threshold. We mitigate confounds by keeping the simulator, rendering pipeline, scene parameterization, evaluation budget, and perception stack fixed across comparisons, and by following recommended configurations for both the generators and YOLOv11.

External validity. We study two pick-and-place scenarios requiring oriented bounding-box detection of cardboard boxes, using Isaac Sim and images collected from two real robotic cells. Confidentiality constraints prevent releasing the full industrial environment and require a representative development setup, which may not capture all real-cell variability. We mitigate this threat by evaluating across two use cases with different box scales and clutter levels, and by adding realistic distractors recommended by robotics engineers to reflect common shop-floor conditions.

Reliability and conclusion validity. STRIM includes stochastic stages (dataset sampling, GAN training, and GA/RS search), which introduce run-to-run variability. We address this by repeating experiments, reporting aggregated statistics with variability, and using Mann–Whitney U tests with Cliff's delta effect sizes to support comparisons.

8.3.4 Data Availability

As part of the replication package, we provide the datasets used to train the generative models, along with the corresponding training instructions and scripts. We also release the data and models required to replicate RQ1, RQ2, and RQ4. The package is available

at `https://figshare.com/s/3d0b811a5f2e49bcea3d`. Due to intellectual property restrictions imposed by our industrial partner, we are unable to release the code required to reproduce the in-simulation test generation within the Isaac Sim environment.

8.4 Chapter Summary

This chapter introduced STRIM, a sim2real-aware framework for in-simulation testing and repair of manipulator vision models that narrows the sim-to-real gap using non-labeled real-world data. STRIM integrates unpaired style transfer to align simulated renderings with the target visual domain and evolutionary search to systematically generate corner-case scenes that expose perception module limitations. The discovered failures are then reused for fine-tuning, closing the loop between testing and repair within a simulation-driven workflow. Across two pick-and-place use cases in NVIDIA Isaac Sim, training on FastCUT-transferred synthetic images improves performance over a simulation-only baseline, yielding F1 gains of roughly 14% to 16% with modest $AP_{50–95}$ improvements. Under the same test budget, GA-based testing discovers more unique failures than random search (+47% and +16%), and fine-tuning on these failures reduces GA-revealed failures by 2.7× and 3.7×. On real-world data, STRIM fine-tuning outperforms the sim-only baseline by up to 3.5% in $AP_{50–95}$ and 17.3% in F1.

CHAPTER 9 CONCLUSIONS AND FUTURE WORK

9.1 Conclusions

This thesis contributes to solving central challenges in the simulation-based testing and improvement of autonomous robotic systems: how to efficiently discover safety-relevant, behaviorally plausible failures in large and complex search spaces, and how to leverage these failures to improve real-world system performance. Considering operational knowledge is indispensable in testing autonomous robotic systems. This knowledge can be expressed in various forms, such as system requirements, custom evaluation functions, surrogate models, safety rules, previously discovered failures or real world data. In this thesis, we propose 5 novel approaches for autonomous RS testing and improvement leveraging operational knowledge across these different dimensions. Considering test generation, we make the following main conclusions:

- **RIGAA** approach proves that learning test generation policies guided by operational knowledge encoded as approximate evaluation functions is beneficial for search-based testing. In experimental evaluation, it allowed revealing from 31% to 37% more failures, compared to the same search without the RL agent. Operational knowledge distilled into a pre-trained reinforcement learning agent can be used to initialize evolutionary search more effectively than random or heuristic-only strategies. By guiding the search toward safety-critical regions of the test space from the outset, RIGAA reduces wasted evaluation effort on irrelevant scenarios while increasing failure discovery rates. The RL agent leverages gradient-based optimization to learn reusable test generation patterns and problem constraints from the given evaluation functions. RL algorithms naturally balance exploration and exploitation, making them well suited for discovering failure-revealing test scenarios. Unlike approaches that rely on pre-existing failures, an RL agent can uncover previously unknown failure cases. Moreover, combining an RL agent with evolutionary search is more effective than using the RL agent alone, as it leads to the generation of more diverse and challenging test scenarios. Evolutionary search further refines the scenarios discovered by the RL agent, producing novel failure-inducing inputs. Finally, unlike greedy initialization strategies, RL agents require a one-time training effort and can be reused across multiple testing campaigns. To sum up, when inexpensive approximate evaluation functions are available, learning test generators using RL is a promising direction, as it enables the discovery of failure cases that might not be identified by evolutionary search alone.

- Search space representation is important for effective search-based testing. In **RILaST** we demonstrated that operational knowledge can be embedded not only in initialization, but also in the representation of the test space itself. By learning latent representations from randomized or operational knowledge optimized test datasets, RILaST enabled search in a transformed space that better captures non-linear dependencies between scenario parameters. This led to substantially higher failure discovery rates and improved diversity, while eliminating the need for handcrafted search operators. Compared to optimization in original space, from 2.4 to 3.5 times more failures were identified in RILaST optimized space. In summary, operational knowledge–based evaluation functions can be used to shape the latent test scenario space, enabling the subsequent search guided by computationally expensive objective functions to be performed more efficiently.

- Considering safety rules is critical for generating avoidable and valid test scenarios in dynamic environments. Signal temporal logic is helpful in automatic analysis of the simulation traces. **Dynasto** leverages operational knowledge of the safety rules, encoded into STL predicates, to guide the RL agent-based behavior generation. Dynasto addresses a critical limitation of prior work by explicitly separating and coordinating optimization of static and dynamic environmental elements. By combining validity-aware reinforcement learning for adversarial behavior synthesis with evolutionary search over initial conditions, Dynasto reveals a broader and more behaviorally realistic set of failure modes. Dynasto identifies 70% unique failures and 50% more failure modes compared to a validity-aware RL baseline. These results confirm that, first, incorporating operational constraints and behavioral validity into dynamic scenario generation is essential for revealing plausible failures and, second, that both static and dynamic element optimization is important in interactive multiagent scenarios.

While the first part of the thesis focused on improving failure discovery, the second part demonstrated how operational knowledge extracted from real world data can be reused to improve system performance. In this part we make the following conclusions:

- High-fidelity simulation environments are valuable for autonomous system testing and improvement. However, the simulation-to-reality gap remains non-negligible even with photorealistic simulators. **MARTENS** demonstrates that failures uncovered through simulation-based testing can be directly leveraged to repair perception models in an industrial robotic manipulator setting. In the experimental evaluation on two use cases, mean average precision (mAP) scores of 0.915 and 0.822 on two real-world test

datasets were achieved from training and fine-tuning on purely simulated data. By additional fine-tuning the models on real-world data collected in the target robotic cell environment, the mAP improves to 0.948 and 0.892, respectively. Simulated data allows to pre-train the model from real world use, however real world data remains critical for further performance improvement. By closing the loop between failure detection, analysis, and fine-tuning, MARTENS enabled real world DNN performance improvements while minimizing additional real-world data collection (since pre-trained model is provided). In this work system level tests were executed and most of the failures originated from the perception component mispredictions. Very limited number of failures were attributed to the control module malfunctions, i.e., two failures for each use case. Given the high computational cost of executing full simulation, we opt for offline DNN testing in the future work.

- Unpaired generative style transfer models are effective in reducing the simulation to reality gap for perception components. Among compared CycleGan, CUT and FastCUT models, FastCUT generative model [65] appeared to be the most efficient in terms of training time and sim2real reduction. **STRIM** combines sim-to-real–aware visual adaptation with search-based corner-case generation and demonstrates that real-world perception performance can be improved using unlabeled real-world data. Specifically, STRIM outperforms a simulation-only baseline by up to 3.5% in $AP_{50–95}$ and 17.3% in F1 score. The results show that operational knowledge about the target deployment domain, encoded through unpaired style transfer and realistic real world environment representation in simulation, can be effectively integrated into simulation-based testing workflows, leading to measurable gains on real data.

Taken together, the contributions of this thesis focus on approaches that maximize the usefulness of operational knowledge embedded in the search while mitigating potential biases from this knowledge. RIGAA introduces reusable models for smart search initialization, and RILaST provides an optimized search space, enhancing the efficiency of failure search when full system simulations are executed. Dynasto enables the generation of valid and avoidable test scenarios by incorporating safety rules and optimizing static and dynamic environmental elements. MARTENS outlines a pipeline for simulation-based testing and improvement using real-world data. STRIM facilitates real-world data integration by reducing labelling effort and increasing the transferability of simulation-found failures to real systems. By smartly integrating operational knowledge into the search process, it becomes possible to explore vast scenario spaces more efficiently, discover more meaningful failures, and directly translate testing outcomes into real-world system improvements.

9.2 Future Work

In this section, we discuss the promising avenues for future work on ARS testing, as well as the limitations of the proposed approaches. This thesis opens several promising research directions aimed at further advancing search-based testing and improvement of autonomous robotic systems. Evident avenue for future work is integration of all the proposed approaches into a unified, closed-loop framework. While RIGAA, RILaST, Dynasto, MARTENS, and STRIM were evaluated independently, they address complementary aspects of the problem. Combining them would enable an end-to-end pipeline in which operational knowledge guides test initialization (RIGAA), shapes the test representation (RILaST), governs valid dynamic behavior generation (Dynasto), and feeds discovered failures back into system repair and adaptation (MARTENS and STRIM).

9.2.1 Test Generation Approaches

Learning domain knowledge through surrogate modelling and uncertainty estimation. In RIGAA and RILaST, we relied on operational-knowledge-based evaluation functions, such as road topology curvature or planning algorithm path length. However, such domain knowledge may not always be available or may not directly apply to a given system under test. Instead of relying on manually defined evaluation functions, we consider it promising to learn these functions directly from the abundant data generated by system executions in simulated environments. To improve scalability to high-dimensional input spaces, as suggested by Haq et al. [24], multiple surrogate models can be trained, each specializing in a different input modality or parameter subset. These models can be constructed offline using supervised learning techniques, or learned online using approaches such as Bayesian optimization with Gaussian processes [276]. Gaussian process–based models enable iterative refinement by actively sampling the most informative regions of the search space. Moreover, they naturally provide uncertainty estimates alongside predictions. Uncertainty estimation is critical for the effective use of surrogate models in the presence of previously unseen inputs. By explicitly quantifying prediction uncertainty, the search process can prioritize inputs that are either highly uncertain or predicted to be high risk, thereby improving both exploration efficiency and failure discovery.

Learning problem representations. In RILaST, we focused on improving existing representations, which requires an initial vectorized representation of the problem to be defined a priori. However, modern deep neural network architectures naturally support multimodal inputs and can directly process heterogeneous data types, such as images, graphs, or 3D objects.

The latent spaces learned through the compression–reconstruction process of these models can therefore serve as learned problem representations. For example, instead of explicitly defining and manipulating object-level features within an image, raw images can be provided as inputs, allowing the representation to be learned implicitly. Such learned representations can significantly simplify the configuration of search-based test generation algorithms and facilitate their application across a broader range of systems and input modalities.

Automatic operational domain identification. In our approaches, we primarily focused on failure identification. From a developer's perspective, however, it is also important to characterize the operational boundaries of a system, beyond which it can no longer reliably complete its intended tasks. For example, identifying the minimum illumination level required for a robotic system to correctly perceive an object, or the minimum distance to a leading vehicle below which the system under test can no longer react adequately to braking events, provides critical insight into system limitations. Addressing such questions requires additional post-processing of failure data, including clustering and root cause analysis. Moreover, it necessitates the definition of metrics that explicitly characterize the operational domain and its boundaries. We leave the automated identification of operational domains as an important direction for future work to be integrated into our test generation approaches.

Multi-fidelity level simulation for test prioritization. High-fidelity simulators provide accurate evaluations of system behavior, but they are computationally expensive and time-consuming to execute. In many cases, however, such high levels of fidelity are not strictly necessary to estimate system behavior with sufficient accuracy. For example, simulating the outputs of certain sensors, such as LiDARs, or the predictions of planning modules does not always require graphics-intensive simulation. Instead, these components can often be approximated using simplified or even purely 2D simulation environments. Test scenarios that appear challenging or failure-prone in these low-fidelity simulators can then be prioritized and re-evaluated using high-fidelity simulation. This multi-fidelity strategy enables more efficient test scenario exploration and may serve as an alternative to approximated evaluation functions. This setup would benefit both MARTENS and STRIM. The perception system could be tested offline in a photorealistic IsaacSim simulator, while the control module execution could be carried out in simpler simulators.

Integration of test generation with real world systems. While a wide range of test generation approaches has been proposed in the literature, their integration with real-world software and hardware systems remains a significant challenge. Bridging this gap requires testing frameworks that explicitly support both software-in-the-loop (SIL) and hardware-in-the-loop (HIL) configurations, providing a seamless integration between test generation

algorithms and real world robotic systems or modules. This challenge was particularly evident during our collaboration with an industrial partner, where the deployment of our testing strategies was complicated by the need to first develop a ROS 2–based framework for inter-component communication [277]. Such integrated testing setups can also contribute to reducing the simulation-to-reality gap at the level of control input execution, enabling more faithful validation of system behavior under realistic operating conditions.

System repair approaches. In our work, notably in MARTENS and STRIM, system repair was achieved by fine-tuning the original DNN using additional data generated through testing. However, this repair process involves several non-trivial design choices, including the size of the fine-tuning dataset and the selection of hyperparameters such as the learning rate and batch size. In our studies, these parameters were selected empirically and guided by recommendations from the existing literature. Nevertheless, more systematic and in-depth investigations of the repair procedure would be beneficial. In particular, when multiple fine-tuning cycles are performed, the risk of catastrophic forgetting becomes increasingly significant. This highlights the need for repair approaches that ensure performance improvements on newly discovered failure cases without degrading the system's previously acquired capabilities.

9.2.2 Test Evaluation Metrics

Test diversity estimation. Test diversity has long been a challenging problem, as diversity metrics often require careful adaptation to the specific characteristics of the test space. In our studies, we represented solutions as one-dimensional normalized vectors, which enabled the use of standard similarity and distance metrics such as cosine and Euclidean distance. However, no large-scale empirical studies have been conducted to systematically analyze which metrics best capture meaningful test diversity, or to determine which metrics are most suitable for different testing problems and input representations. Moreover, existing works often report the number of failures revealed and their diversity as two separate metrics, which makes direct comparison between approaches challenging. We advocate for the development of unified metrics, such as the number of unique failures or distinct failure modes discovered, that simultaneously capture both the failure-revealing power and the diversity preservation capability of a given approach.

Failure mode clustering. Failure clustering is a widely used step for the automated analysis and interpretation of test results. Existing approaches in the literature employ a variety of clustering algorithms, including k-means, DBSCAN, hierarchical clustering, and community detection methods. Based on our experimental observations, these algorithms are highly

sensitive to hyperparameter choices and can produce substantially different clusterings under different configurations. In Dynasto, we address this limitation by performing a calibration on a manually selected set of test scenarios. Despite the widespread adoption of clustering, large-scale comparative studies that evaluate and recommend clustering techniques based on problem characteristics and failure representations remain absent.

Universal signal temporal logic rules and frameworks for interactive scenarios. In Dynasto, we relied on manually defined signal temporal logic rules derived from established safety requirements and standards. We argue that there is a need for the systematic definition, standardization, and reuse of such rules within the autonomous robotic systems testing community. Shared STL rule libraries could serve as reusable oracles for multi-agent and interactive scenarios, where complex interactions may lead to collisions or other safety violations, and where it is necessary to determine whether a failure was avoidable (i.e., valid) or inevitable.

9.2.3 Benchmarks

System under test benchmarks. In our work, we considered a limited number of systems under test due to the scarcity of reusable benchmarks and the high cost of setting one up. Future research should be able to explore extending the proposed techniques to more complex multi-agent and multi-modal settings, including richer perception tasks, longer-horizon planning problems, and interactive environments with multiple autonomous agents. Applying the developed approach to additional domains—such as collaborative robotics, aerial systems, and safety-critical industrial automation should be easily accessible and configurable. Moreover, software engineering and robotics communities should collaborate more closely to make state-of-the art robotics systems easily testable using established software engineering methodologies and tools.

Simulation to reality minimization benchmarks. The simulation-to-reality (sim-to-real) gap remains a central challenge in search-based testing conducted in simulation. This gap can be mitigated using techniques such as generative style transfer models. However, the availability of benchmark datasets for evaluating sim-to-real transfer remains limited. Moreover, most existing datasets and evaluations are concentrated in the autonomous driving domain. In STRIM, we manually constructed and open-sourced two datasets from robotic manipulation domains, which can serve as benchmarks for evaluating style transfer models beyond self-driving scenarios. Different style transfer methods can exhibit different performance depending on the target domain. We therefore advocate for the creation of multi-domain sim-to-real benchmarks and large-scale comparative evaluations of available style

transfer models, to better guide software test engineers in selecting the most appropriate technique for their specific application contexts.

REFERENCES

[1] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[2] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ: Pearson, 2021.

[3] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[4] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[5] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Springer, 2016.

[6] J. Albright, J. Schneider, and C. Nyce, "The chaotic middle: The autonomous vehicle and disruption in automobile insurance," 2017.

[7] R. R. Murphy, *Disaster Robotics*. MIT Press, 2014.

[8] N. Krüger *et al.*, "Robotics in hazardous environments," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, pp. 1–24, 2019.

[9] E. E. Alves, D. Bhatt, B. Hall, K. Driscoll, A. Murugesan, and J. Rushby, "Considerations in assuring safety of increasingly autonomous systems," 2018.

[10] P. I. Corke and O. Khatib, *Robotics, vision and control: fundamental algorithms in MATLAB*. Springer, 2011, vol. 73.

[11] National Transportation Safety Board (NTSB), "Collision between vehicle controlled by developmental automated driving system and pedestrian," National Transportation Safety Board, Tech. Rep. NTSB/HAR-19/03, 2019, accessed: 2025-12-29. [Online]. Available: https://www.ntsb.gov/investigations/AccidentReports/Reports/HAR1903.pdf

[12] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, 2017.

[13] C. Badue *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165, p. 113816, 2021.

[14] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[15] B. R. Kiran *et al.*, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2022.

[16] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza, "Champion-level drone racing using deep reinforcement learning," *Nature*, vol. 620, no. 7976, pp. 982–987, 2023.

[17] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "Transfuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 11, pp. 12 878–12 895, 2022.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] CBS News. (2022) Two motorcyclists dead in crashes involving tesla vehicles using autopilot, nhtsa says. CBS News. Accessed: 2025-12-29. [Online]. Available: https://www.cbsnews.com/news/tesla-crashes-killed-2-motorcyclists-autopilot-nhtsa/

[20] ——. (2022) Two motorcyclists dead in crashes involving tesla vehicles using autopilot, nhtsa says. CBS News. Accessed: 2025-12-29. [Online]. Available: https://www.cbsnews.com/news/tesla-crashes-killed-2-motorcyclists-autopilot-nhtsa/

[21] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation research part A: policy and practice*, vol. 94, pp. 182–193, 2016.

[22] C. Birchler, S. Khatiri, P. Rani, T. Kehrer, and S. Panichella, "A roadmap for simulation-based testing of autonomous cyber-physical systems: Challenges and future direction," *ACM Transactions on Software Engineering and Methodology*, vol. 34, no. 5, pp. 1–9, 2025.

[23] J. M. Whitt and P. J. Bounker, "The use and benefits of modeling and simulation with autonomous vehicles," SAE Technical Paper, Tech. Rep., 2024.

[24] F. U. Haq, D. Shin, and L. Briand, "Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 811–822.

[25] G. Jahangirova and P. Tonella, "An empirical evaluation of mutation operators for deep learning systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 74–84.

[26] F. Klück, M. Zimmermann, F. Wotawa, and M. Nica, "Genetic algorithm-based test parameter optimization for adas system testing," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 418–425.

[27] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I 31*. Springer, 2019, pp. 432–442.

[28] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: a replication and extension study," *Empirical Software Engineering*, vol. 28, no. 3, p. 73, 2023.

[29] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 384–402, 2022.

[30] A. Doreste, M. Biagiola, and P. Tonella, "Adversarial testing with reinforcement learning: A case study on autonomous driving," in *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2024, pp. 293–304.

[31] M. H. Amini and S. Nejati, "Bridging the gap between real-world and synthetic images for testing autonomous driving systems," in *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2024, pp. 732–744.

[32] NVIDIA, "Nvidia isaac sim: Photorealistic simulation for robotics and autonomous systems," https://developer.nvidia.com/isaac-sim, 2023, accessed: 2026-01-XX.

[33] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017.

[34] H. Fahmy, F. Pastore, L. Briand, and T. Stifter, "Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, pp. 1–47, 2023.

[35] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 143–154.

[36] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1814–1826.

[37] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, Dec. 2012. [Online]. Available: https://doi.org/10.1145/2379776.2379787

[38] A. Gambi, M. Mueller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 318–328.

[39] R. B. Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 1016–1026.

[40] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[41] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

[42] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 69–93.

[43] G. Antoniol, M. Di Penta, and M. Harman, "Search-based techniques applied to optimization of project planning for a massive maintenance project," in *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005, pp. 240–249.

[44] D. Dumitrescu, B. Lazzerini, L. C. Jain, and A. Dumitrescu, *Evolutionary computation*. CRC press, 2000.

[45] C. A. C. Coello, G. B. Lamont, D. A. Van Veldhuizen *et al.*, *Evolutionary algorithms for solving multi-objective problems*. Springer, 2007, vol. 5.

[46] S. Verma, M. Pant, and V. Snasel, "A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems," *Ieee Access*, vol. 9, pp. 57 757–57 791, 2021.

[47] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[48] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[50] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE signal processing magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[51] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[52] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[53] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *International Conference on Learning Representations (ICLR)*, 2014.

[54] J. Adler and S. Lunz, "Banach wasserstein gan," *Advances in neural information processing systems*, vol. 31, 2018.

[55] D. P. Kingma, M. Welling *et al.*, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019.

[56] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[57] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[58] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," *IEEE Symposium Series on Computational Intelligence*, 2020.

[59] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[60] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[61] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model: A review," *Neural Networks*, vol. 69, pp. 60–79, 2015.

[62] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[63] K. Bousmalis, A. Irpan, P. Wohlhart *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[64] M. Biagiola, A. Stocco, V. Riccio, and P. Tonella, "Two is better than one: digital siblings to improve autonomous driving testing," *Empirical Software Engineering*, vol. 29, no. 4, p. 72, 2024.

[65] T. Park, A. A. Efros, R. Zhang, and J.-Y. Zhu, "Contrastive learning for unpaired image-to-image translation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[66] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[67] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[68] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[69] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 254–257.

[70] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 167–170.

[71] C. Menghi, S. Nejati, L. Briand, and Y. I. Parache, "Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 2020, pp. 372–384.

[72] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," *Empirical Software Engineering and Verification: International Summer Schools, LASER 2008-2010, Elba Island, Italy, Revised Tutorial Lectures*, pp. 1–59, 2012.

[73] F. Formica, T. Fan, and C. Menghi, "Search-based software testing driven by automatically generated and manually defined fitness functions," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, pp. 1–37, 2023.

[74] F. Formica, M. Lawford, and C. Menghi, "Search-based software testing driven by domain knowledge: Reflections and new perspectives," *arXiv preprint arXiv:2512.10079*, 2025.

[75] A. Arrieta, J. A. Agirre, and G. Sagardui, "A tool for the automatic generation of test cases and oracles for simulation models based on functional requirements," in *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2020, pp. 1–5.

[76] Y. Luo, X.-Y. Zhang, P. Arcaini, Z. Jin, H. Zhao, F. Ishikawa, R. Wu, and T. Xie, "Targeting requirements violations of autonomous driving systems by dynamic evolutionary search," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 279–291.

[77] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Deephyperion: Exploring the feature space of deep learning-based systems through illumination search," 2021.

[78] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1555–1562.

[79] P. Arcaini and A. Cetinkaya, "Crag–a combinatorial testing-based generator of road geometries for ads testing," *Science of Computer Programming*, vol. 238, p. 103171, 2024.

[80] D. Humeniuk, G. Antoniol, and F. Khomh, "Ambiegen tool at the sbst 2022 tool competition," in *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2022, pp. 43–46.

[81] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

[82] E. Castellano, A. Cetinkaya, C. H. Thanh, S. Klikovits, X. Zhang, and P. Arcaini, "Frenetic at the sbst 2021 tool competition," in *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2021, pp. 36–37.

[83] A. A. Babikian and D. Varró, "Optangle at the sbft 2024 tool competition-cyber-physical systems track," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, 2024, pp. 73–74.

[84] S. Khatiri, S. Panichella, and P. Tonella, "Simulation-based testing of unmanned aerial vehicles with aerialist," in *International Conference on Software Engineering (ICSE)*, 2024.

[85] S. Khatiri, P. Saurabh, T. Zimmermann, C. Munasinghe, C. Birchler, and S. Panichella, "Sbft tool competition 2024: Cps-uav test case generation track," in *17th International Workshop on Search-Based and Fuzz Testing (SBFT), Lisbon, Portugal, 14-20 April 2024*. ZHAW Zürcher Hochschule für Angewandte Wissenschaften, 2024.

[86] M. D. Liso and Z. W. Soi, "Camba cps-uav at the sbft tool competition 2024," in *IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT@ICSE 2024*, 2024.

[87] S. Tang, Z. Zhang, A. Cetinkaya, and P. Arcaini, "Palm at the icst 2025 tool competition–uav testing track," in *2025 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2025, pp. 823–824.

[88] Z. Jiang and A. A. Babikian, "Optobstacles at the sbft 2025 tool competition–uav testing track," in *2025 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE, 2025, pp. 1–2.

[89] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia, "Doppelgänger test generation for revealing bugs in autonomous driving software."

[90] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints," *IEEE transactions on evolutionary computation*, vol. 18, no. 4, pp. 577–601, 2013.

[91] Y. Huai, S. Almanee, Y. Chen, X. Wu, Q. A. Chen, and J. Garcia, "sceno rita: Generating diverse, fully-mutable, test scenarios for autonomous vehicle planning," *IEEE Transactions on Software Engineering*, 2023.

[92] S. Kim, M. Liu, J. J. Rhee, Y. Jeon, Y. Kwon, and C. H. Kim, "Drivefuzz: Discovering autonomous driving bugs through driving quality-guided fuzzing," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1753–1767.

[93] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer, "Av-fuzzer: Finding safety violations in autonomous driving systems," in *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE, 2020, pp. 25–36.

[94] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter, "Testing advanced driver assistance systems using multi-objective search and neural networks," in *Proceedings of the 31st IEEE/ACM international conference on automated software engineering*, 2016, pp. 63–74.

[95] L. Sorokin and N. Kerscher, "Guiding the search towards failure-inducing test inputs using support vector machines," in *Proceedings of the 5th IEEE/ACM International Workshop on Deep Learning for Testing and Testing for Deep Learning*, 2024, pp. 9–12.

[96] C. Birchler, N. Ganz, S. Khatiri, A. Gambi, and S. Panichella, "Cost-effective simulation-based test selection in self-driving cars software with sdc-scissor," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 164–168.

[97] T. Zohdinasab, V. Riccio, and P. Tonella, "Focused test generation for autonomous driving systems," *ACM Transactions on Software Engineering and Methodology*, 2024.

[98] A. Zolfagharian, M. Abdellatif, L. C. Briand, M. Bagherzadeh, and S. Ramesh, "A search-based testing approach for deep reinforcement learning agents," *IEEE Transactions on Software Engineering*, 2023.

[99] J. Peltomäki and I. Porres, "Learning test generators for cyber-physical systems," *arXiv preprint arXiv:2410.03202*, 2024.

[100] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*. PMLR, 2017, pp. 214–223.

[101] J. Winsten, V. Soloviev, J. Peltomäki, and I. Porres, "Adaptive test generation for unmanned aerial vehicles using wogan-uav," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, 2024, pp. 43–44.

[102] J. Peltomäki and I. Porres, "Requirement falsification for cyber-physical systems using generative models," *Automated Software Engineering*, vol. 32, no. 2, pp. 1–48, 2025.

[103] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," in *Handbook of statistics*. Elsevier, 2013, vol. 31, pp. 35–59.

[104] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," *Advances in neural information processing systems*, vol. 31, 2018.

[105] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 63–78.

[106] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer, 2011, pp. 507–523.

[107] S. Ramakrishna, B. Luo, C. B. Kuhn, G. Karsai, and A. Dubey, "Anti-carla: An adversarial testing framework for autonomous vehicles in carla," in *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2022, pp. 2620–2627.

[108] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Conference on robot learning*. PMLR, 2017, pp. 1–16.

[109] S. Ramakrishna, B. Luo, Y. Barve, G. Karsai, and A. Dubey, "Risk-aware scene sampling for dynamic assurance of autonomous systems," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*. IEEE, 2022, pp. 107–116.

[110] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 1443–1450.

[111] E. Snelson, "Tutorial: Gaussian process models for machine learning," *Gatsby Computational Neuroscience Unit, UCL*, 2006.

[112] M. H. Moghadam, M. Borg, M. Saadatmand, S. J. Mousavirad, M. Bohlin, and B. Lisper, "Machine learning testing in an adas case study using simulation-integrated bio-inspired search-based testing," *arXiv preprint arXiv:2203.12026*, 2022.

[113] G. Fraser and A. Arcuri, "The seed is strong: Seeding strategies in search-based software testing," in *2012 IEEE fifth international conference on software testing, verification and validation*. IEEE, 2012, pp. 121–130.

[114] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, A. Egyed, and E. Alba, "Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines," in *2014 IEEE congress on evolutionary computation (CEC)*. IEEE, 2014, pp. 387–396.

[115] A. Arrieta, P. Valle, J. A. Agirre, and G. Sagardui, "Some seeds are strong: Seeding strategies for search-based test case selection," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, pp. 1–47, 2023.

[116] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making: 9th Asian Computing*

Science Conference. Dedicated to Jean-Louis Lassez on the Occasion of His 5th Birthday. Chiang Mai, Thailand, December 8-10, 2004. Proceedings 9. Springer, 2005, pp. 320–329.

[117] A. Gambi, V. Nguyen, J. Ahmed, and G. Fraser, "Generating critical driving scenarios from accident sketches," in *2022 IEEE International Conference On Artificial Intelligence Testing (AITest).* IEEE, 2022, pp. 95–102.

[118] S. Khatiri, S. Panichella, and P. Tonella, "Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights," in *2023 IEEE Conference on Software Testing, Verification and Validation (ICST).* IEEE, 2023, pp. 281–292.

[119] F. Scheuer, A. Gambi, and P. Arcaini, "Stretch: Generating challenging scenarios for testing collision avoidance systems," in *2023 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2023, pp. 1–6.

[120] E. Castellano, A. Cetinkaya, and P. Arcaini, "Analysis of road representations in search-based testing of autonomous driving systems," in *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS).* IEEE, 2021, pp. 167–178.

[121] K. Tapp, *Differential geometry of curves and surfaces.* Springer, 2016.

[122] S. Tang, Z. Zhang, J. Zhou, Y. Zhou, Y.-F. Li, and Y. Xue, "Evoscenario: Integrating road structures into critical scenario generation for autonomous driving system testing," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE).* IEEE, 2023, pp. 309–320.

[123] C. Lu, S. Ali, and T. Yue, "Epitester: Testing autonomous vehicles with epigenetic algorithm and attention mechanism," *IEEE Transactions on Software Engineering*, 2024.

[124] D. H. Stolfi and E. Alba, "Epigenetic algorithms: A new way of building gas based on epigenetics," *Inf. Sci.*, vol. 424, no. C, p. 250–272, Jan. 2018. [Online]. Available: https://doi.org/10.1016/j.ins.2017.10.005

[125] A. Gaier, A. Asteroth, and J.-B. Mouret, "Discovering representations for black-box optimization," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 103–111.

[126] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," 2015.

[127] P. J. Bentley, S. L. Lim, A. Gaier, and L. Tran, "Coil: Constrained optimization in learned latent space: learning representations for valid solutions," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2022, pp. 1870–1877.

[128] P. Bentley, S. L. Lim, P. Arcaini, and F. Ishikawa, "Using a variational autoencoder to learn valid search spaces of safely monitored autonomous robots for last-mile delivery," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023, pp. 1303–1311.

[129] P. J. Bentley, S. L. Lim, A. Gaier, and L. Tran, "Evolving through the looking glass: Learning improved search spaces with variational autoencoders," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2022, pp. 371–384.

[130] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 2015, pp. 6C2–1.

[131] R. Lee, O. J. Mengshoel, A. Saksena, R. W. Gardner, D. Genin, J. Silbermann, M. Owen, and M. J. Kochenderfer, "Adaptive stress testing: Finding likely failure events with reinforcement learning," *Journal of Artificial Intelligence Research*, vol. 69, pp. 1165–1201, 2020.

[132] H. Delecki, M. Itkina, B. Lange, R. Senanayake, and M. J. Kochenderfer, "How do we fail? stress testing perception in autonomous vehicles," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 5139–5146.

[133] D. Karunakaran, S. Worrall, and E. Nebot, "Efficient statistical validation with edge cases to evaluate highly automated vehicles," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–8.

[134] B. Chen, X. Chen, Q. Wu, and L. Li, "Adversarial evaluation of autonomous vehicles in lane-change scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 10 333–10 342, 2021.

[135] F. Bella and R. Russo, "A collision warning system for rear-end collision: a driving simulator study," *Procedia-social and behavioral sciences*, vol. 20, pp. 676–686, 2011.

[136] Z. Li, J. Dai, Z. Huang, N. You, Y. Zhang, and M. Yang, "Viohawk: Detecting traffic violations of autonomous driving systems through criticality-guided simulation test-

ing," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 844–855.

[137] A. Stocco, B. Pulfer, and P. Tonella, "Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2022.

[138] L. Sorokin, M. Biagiola, and A. Stocco, "Simulator ensembles for trustworthy autonomous driving testing," *arXiv preprint arXiv:2503.08936*, 2025.

[139] L. Baresi, D. Y. X. Hu, A. Stocco, and P. Tonella, *Efficient Domain Augmentation for Autonomous Driving Testing Using Diffusion Models*. IEEE Press, 2025, p. 398–410. [Online]. Available: https://doi.org/10.1109/ICSE55347.2025.00206

[140] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems," *Empirical Software Engineering*, vol. 26, no. 5, p. 90, 2021.

[141] B. Yu, H. Qi, Q. Guo, F. Juefei-Xu, X. Xie, L. Ma, and J. Zhao, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, vol. 71, no. 4, pp. 1401–1416, 2021.

[142] X. Gao, R. K. Saha, M. R. Prasad, and A. Roychoudhury, "Fuzz testing based data augmentation to improve robustness of deep neural networks," in *Proceedings of the acm/ieee 42nd international conference on software engineering*, 2020, pp. 1147–1158.

[143] Z. Li, M. Pan, T. Zhang, and X. Li, "Testing dnn-based autonomous driving systems under critical environmental conditions," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6471–6482.

[144] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal unsupervised image-to-image translation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 172–189.

[145] M. O. Attaoui, F. Pastore, and L. Briand, "Search-based dnn testing and retraining with gan-enhanced simulations," *arXiv preprint arXiv:2406.13359*, 2024.

[146] A. R. Plummer, "Model-in-the-loop testing," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 220, no. 3, pp. 183–199, 2006.

[147] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE '20. Association for Computing Machinery, 2020, p. 13 pages.

[148] BeamNG.tech, "Beamng gmbh." 2021. [Online]. Available: https://www.beamng.gmbh/research

[149] G. Briffoteaux, "Parallel surrogate-based algorithms for solving expensive optimization problems," Ph.D. dissertation, Université de Lille; Université de Mons (UMONS), 2022.

[150] A. Rosales-Pérez, C. A. C. Coello, J. A. Gonzalez, C. A. Reyes-Garcia, and H. J. Escalante, "A hybrid surrogate-based approach for evolutionary multi-objective optimization," in *2013 IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 2548–2555.

[151] J. M. Rojas, G. Fraser, and A. Arcuri, "Seeding strategies in search-based unit test generation," *Software Testing, Verification and Reliability*, vol. 26, no. 5, pp. 366–401, 2016.

[152] A. Arcuri, D. R. White, J. Clark, and X. Yao, "Multi-objective improvement of software using co-evolution and smart seeding," in *Simulated Evolution and Learning: 7th International Conference, SEAL 2008, Melbourne, Australia, December 7-10, 2008. Proceedings 7*. Springer, 2008, pp. 61–70.

[153] T. Chen, M. Li, and X. Yao, "Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition," *Information and Software Technology*, vol. 114, pp. 155–175, 2019.

[154] K. Mao, M. Harman, and Y. Jia, "Sapienz: Multi-objective automated testing for android applications," in *Proceedings of the 25th international symposium on software testing and analysis*, 2016, pp. 94–105.

[155] A. Gambi, G. Jahangirova, V. Riccio, and F. Zampetti, "Sbst tool competition 2022," in *2022 IEEE/ACM 15th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2022, pp. 25–32.

[156] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2020.

[157] D. Humeniuk, F. Khomh, and G. Antoniol, "A search-based framework for automatic generation of testing environments for cyber-physical systems," *Information and Software Technology*, p. 106936, 2022.

[158] J. Arnold and R. Alexander, "Testing autonomous robot control software using procedural content generation," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2013, pp. 33–44.

[159] T. Sotiropoulos, G. Guiochet, I. Ingrand, and W. Waeselynck, "Virtual worlds for testing robot navigation: a study on the difficulty level," in *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 2016, pp. 153–160.

[160] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "Sbst tool competition 2021," in *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*. IEEE, 2021, pp. 20–27.

[161] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 42, no. 1, pp. 55–61, 2000.

[162] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Pearson, 2016.

[163] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[164] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8806–8813.

[165] G. Campion, G. Bastin, and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *IEEE transactions on robotics and automation*, vol. 12, no. 1, pp. 47–62, 1996.

[166] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[167] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[168] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 15, no. 9, pp. 850–863, 1993.

[169] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[170] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[171] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 561–591. [Online]. Available: https://proceedings.mlr.press/v87/mahmood18a.html

[172] Y. Wang, H. He, and X. Tan, "Truly proximal policy optimization," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 113–122.

[173] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. AraÃºjo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[174] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[175] M. Harman and B. F. Jones, "Search-based software engineering," *Information and software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[176] B. Hoxha, H. Bach, H. Abbas, A. Dokhanchi, Y. Kobayashi, and G. Fainekos, "Towards formal specification visualization for testing and monitoring of cyber-physical systems," in *Int. Workshop on Design and Implementation of Formal Tools and Systems*. sn, 2014.

[177] N. C. Chung, B. Miasojedow, M. Startek, and A. Gambin, "Jaccard/tanimoto similarity test and estimation methods for biological presence-absence data," *BMC bioinformatics*, vol. 20, no. 15, pp. 1–11, 2019.

[178] J. Lehman and K. O. Stanley, "Evolving a diversity of virtual creatures through novelty search and local competition," in *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, 2011, pp. 211–218.

[179] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[180] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[181] M. Biagiola, S. Klikovits, J. Peltomäki, and V. Riccio, "Sbft tool competition 2023-cyber-physical systems track," in *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE, 2023, pp. 45–48.

[182] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[183] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[184] J. Winsten and I. Porres, "Wogan at the sbft 2023 tool competition-cyber-physical systems track," in *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE, 2023, pp. 43–44.

[185] D. Humeniuk, F. Khomh, and G. Antoniol, "Ambiegen: A search-based framework for autonomous systems testing," *arXiv preprint arXiv:2301.01234*, 2023.

[186] P. Arcaini and A. Cetinkaya, "Crag at the sbft 2023 tool competition-cyber-physical systems track," in *2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT)*. IEEE, 2023, pp. 41–42.

[187] N. Beume, B. Naujoks, and M. Emmerich, "Sms-emoa: Multiobjective selection based on dominated hypervolume," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1653–1669, 2007.

[188] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[189] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[190] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*. PMLR, 2018, pp. 1407–1416.

[191] F. foundation, "Ant gym," 2020. [Online]. Available: https://gymnasium.farama.org/environments/mujoco/ant/

[192] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[193] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *LASER Summer School on Software Engineering*. Springer, 2008, pp. 1–59.

[194] D. Humeniuk, F. Khomh, and G. Antoniol, "Reinforcement learning informed evolutionary search for autonomous system testing," Aug. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8242223

[195] Y. Huai, Y. Chen, S. Almanee, T. Ngo, X. Liao, Z. Wan, Q. A. Chen, and J. Garcia, "Doppelgänger test generation for revealing bugs in autonomous driving software," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2591–2603.

[196] V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, "Evolving mario levels in the latent space of a deep convolutional generative adversarial network," in *Proceedings of the genetic and evolutionary computation conference*, 2018, pp. 221–228.

[197] P. Bontrager, W. Lin, J. Togelius, and S. Risi, "Deep interactive evolution," in *Computational Intelligence in Music, Sound, Art and Design: 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings*. Springer, 2018, pp. 267–282.

[198] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[199] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[200] F. Rothlauf and F. Rothlauf, *Representations for genetic and evolutionary algorithms*. Springer, 2002.

[201] D. Humeniuk and F. Khomh, "Representation improvement in latent space for search based testing of autonomous robotic systems - replication package," Mar. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15087028

[202] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2010, pp. 92–106.

[203] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[204] S. Khatiri, S. Panichella, and P. Tonella, "Simulation-based testing of unmanned aerial vehicles with aerialist," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 134–138.

[205] A. A. Babikian, O. Semeráth, and D. Varró, "Concretization of abstract traffic scene specifications using metaheuristic search," *IEEE Transactions on Software Engineering*, vol. 50, no. 1, pp. 48–68, 2023.

[206] J. Lehman, K. O. Stanley *et al.*, "Exploiting open-endedness to solve problems through the search for novelty." in *ALIFE*, 2008, pp. 329–336.

[207] Q. Zhang and H. Li, "Moea/d: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on evolutionary computation*, vol. 11, no. 6, pp. 712–731, 2007.

[208] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[209] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review." *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[210] S. Klikovits, E. Castellano, A. Cetinkaya, and P. Arcaini, "Frenetic-lib: An extensible framework for search-based generation of road structures for ads testing," *Science of Computer Programming*, vol. 230, p. 102996, 2023.

[211] J. Foreman, "Cosine distance, cosine similarity, angular cosine distance, angular cosine similarity," 2014.

[212] K. Deb and H.-G. Beyer, "Self-adaptive genetic algorithms with simulated binary crossover," *Evolutionary computation*, vol. 9, no. 2, pp. 197–221, 2001.

[213] K. Deb, K. Sindhya, and T. Okabe, "Self-adaptive simulated binary crossover for real-parameter optimization," in *Proceedings of the 9th annual conference on genetic and evolutionary computation*, 2007, pp. 1187–1194.

[214] S. Tang, Z. Zhang, A. Cetinkaya, and P. Arcaini, "Tumb at the sbft 2024 tool competition-cps-uav test case generation track," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, 2024, pp. 53–54.

[215] D. Humeniuk, F. Khomh, and G. Antoniol, "Reinforcement learning informed evolutionary search for autonomous systems testing," *ACM Trans. Softw. Eng. Methodol.*, jul 2024, just Accepted. [Online]. Available: https://doi.org/10.1145/3680468

[216] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[217] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 6235–6240.

[218] D. Humeniuk and F. Khomh, "Ambiegen at the sbft 2024 tool competition-cps-uav track," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, 2024, pp. 69–70.

[219] S. Khatiri, P. Saurabh, T. Zimmermann, C. Munasinghe, C. Birchler, and S. Panichella, "Sbft tool competition 2024 - cps-uav test case generation track," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, ser. SBFT '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 29–32. [Online]. Available: https://doi.org/10.1145/3643659.3643931

[220] M. Biagiola and S. Klikovits, "Sbft tool competition 2024 - cyber-physical systems track," in *Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing*, ser. SBFT '24. New York, NY, USA:

Association for Computing Machinery, 2024, p. 33–36. [Online]. Available: https://doi.org/10.1145/3643659.3643932

[221] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, "Don't decay the learning rate, increase the batch size," *arXiv preprint arXiv:1711.00489*, 2017.

[222] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[223] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[224] J. Han, J. Pei, and H. Tong, *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[225] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 375–386.

[226] H. Ebadi, M. H. Moghadam, M. Borg, G. Gay, A. Fontes, and K. Socha, "Efficient and effective generation of test cases for pedestrian detection-search-based software testing of baidu apollo in svl," in *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 2021, pp. 103–110.

[227] V. A. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, no. 1, p. 5233, 2019.

[228] E. Leurent, "An environment for autonomous driving decision-making," https://github.com/eleurent/highway-env, 2018.

[229] "Iso 34502:2022 - road vehicles - scenario-based safety evaluation framework for automated driving systems," International Organization for Standardization, Geneva, Switzerland, 2022, accessed: 2025-01-XX. [Online]. Available: https://www.iso.org/standard/78951.html

[230] T. Yamaguchi, B. Hoxha, and D. Ničković, "Rtamt–runtime robustness monitors with application to cps and robotics," *International Journal on Software Tools for Technology Transfer*, vol. 26, no. 1, pp. 79–99, 2024.

[231] P. Seiler, B. Song, and J. K. Hedrick, "Development of a collision avoidance system," *SAE transactions*, pp. 1334–1340, 1998.

[232] Entrapeer, "Autonomous driving projects," https://entrapeer.com/market-research/autonomous-vehicles/, Sep. 2023, market research article, posted September 29, 2023. [Online]. Available: https://entrapeer.com/market-research/autonomous-vehicles/

[233] J. Reimann, N. Mansion, J. Haydon, B. Bray, A. Chattopadhyay, S. Sato, M. Waga, É. André, I. Hasuo, N. Ueda *et al.*, "Temporal logic formalisation of iso 34502 critical scenarios: modular construction with the rss safety distance," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 186–195.

[234] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[235] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[236] K. Deb and D. Deb, "Analysing mutation schemes for real-parameter genetic algorithms," *International Journal of Artificial Intelligence and Soft Computing*, vol. 4, no. 1, pp. 1–28, 2014.

[237] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.

[238] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[239] M. O. Attaoui, H. Fahmy, F. Pastore, and L. Briand, "Supporting safety analysis of image-processing dnns through clustering-based approaches," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 5, pp. 1–48, 2024.

[240] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[241] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[242] H. B. Braiek and F. Khomh, "Deepevolution: A search-based testing approach for deep neural networks," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2019, pp. 454–458.

[243] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 132–142.

[244] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Comparing offline and online testing of deep neural networks: An autonomous car case study," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 2020, pp. 85–95.

[245] J. G. Adigun, T. P. Huck, M. Camilli, and M. Felderer, "Risk-driven online testing and test case diversity analysis for ml-enabled critical systems," in *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2023, pp. 344–354.

[246] NVIDIA, "Different types of light sources in nvidia isaac sim," 2022, accessed: 2024-07-15. [Online]. Available: https://omniverse-content-production.s3-us-west-2.amazonaws.com/Assets/Isaac/Documentation/Isaac-Sim-Docs_2022.2.1/materials-and-rendering/latest/103/lighting.html

[247] C. Birchler, S. Khatiri, P. Derakhshanfar, S. Panichella, and A. Panichella, "Single and multi-objective test cases prioritization for self-driving cars in virtual environments," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–30, 2023.

[248] NVIDIA, "Omniverse replicator api documentation," 2023, accessed: 2024-07-15. [Online]. Available: https://docs.omniverse.nvidia.com/py/replicator/1.5.1/source/extensions/omni.replicator.core/docs/API.html

[249] "Kinova gen3 robot." [Online]. Available: https://www.kinovarobotics.com/product/gen3-robots

[250] Robotiq, "2f-85 and 2f-140 adaptive robot grippers," 2023, accessed: 2024-07-15. [Online]. Available: https://robotiq.com/products/2f85-140-adaptive-robot-gripper

[251] G. J. Monkman, S. Hesse, R. Steinmann, and H. Schunk, *Robot grippers*. John Wiley & Sons, 2007.

[252] Intel, "Intel realsense depth camera d435," 2023, accessed: 2024-07-15. [Online]. Available: https://www.intelrealsense.com/depth-camera-d435/

[253] C.-A. Cheng, M. Mukadam, J. Issac, S. Birchfield, D. Fox, B. Boots, and N. Ratliff, "Rmp flow: A computational graph for automatic motion policy generation," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 441–457.

[254] K. Robotics, "Kinova kortex api documentation," 2023, accessed: 2024-07-15. [Online]. Available: https://docs.kinovarobotics.com/

[255] Ultralytics, "Yolov8," 2023, accessed: 2024-07-15. [Online]. Available: https://yolov8.com/

[256] ——, "Yolov8: Oriented bounding boxes," 2023, accessed: 2024-07-15. [Online]. Available: https://yolov8.com/

[257] ——, "Yolov8 training documentation," 2023, accessed: 2024-07-15. [Online]. Available: https://docs.ultralytics.com/modes/train/

[258] A. W. Services, "Amazon ec2 instance types - g5 instances," 2023, accessed: 2024-07-15. [Online]. Available: https://aws.amazon.com/ec2/instance-types/g5/

[259] P. Good, *Permutation tests: a practical guide to resampling methods for testing hypotheses*. Springer Science & Business Media, 2013.

[260] D. Humeniuk, "MARTENS framework: examples of DNN mispredictions discovered for the two use cases," Jul. 2024. [Online]. Available: https://doi.org/10.5281/zenodo.12748885

[261] A. Singh, V. Kalaichelvi, and R. Karthikeyan, "A survey on vision guided robotic systems with intelligent control strategies for autonomous tasks," *Cogent Engineering*, vol. 9, no. 1, p. 2050020, 2022.

[262] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 1316–1322.

[263] "Nvidia isaac sim documentation," https://docs.isaacsim.omniverse.nvidia.com/, accessed: 2025-12-22.

[264] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2019.

[265] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[266] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Pytorch-cyclegan-and-pix2pix," https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix?tab=readme-ov-file, 2020, accessed: 2025-12-22.

[267] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[268] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *CVPR*, 2017.

[269] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[270] M. Bińkowski, D. J. Sutherland, M. Arbel, and A. Gretton, "Demystifying mmd gans," in *International Conference on Learning Representations (ICLR)*, 2018.

[271] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[272] T. Salimans *et al.*, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[273] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[274] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, 2012.

[275] O. Patashnik, D. Danon, H. Zhang, and D. Cohen-Or, "Image translation between imbalanced domains via cross-modal transfer (balagan)," in *CVPR Workshops*, 2021.

[276] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[277] S. Macenski, T. Foote, B. Gerkey, F. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

APPENDIX A LIST OF PUBLICATIONS

The contributions of this thesis include a number of accepted and submitted publications, which are presented below:

Journal publications:

- *Humeniuk, D., Khomh, F. and Antoniol, G., 2023. Ambiegen: A search-based framework for autonomous systems testing. Science of Computer Programming, 230, p.102990.*
- *Humeniuk, D., Khomh, F. and Antoniol, G., 2024. Reinforcement learning informed evolutionary search for autonomous systems testing. ACM Transactions on Software Engineering and Methodology, 33(8), pp.1-45.*
- *Humeniuk, D. and Khomh, F., 2025. Representation Improvement in Latent Space for Search-Based Testing of Autonomous Robotic Systems. ACM Transactions on Software Engineering and Methodology, (under revision).*

Conference publications:

- *Humeniuk, D., Ben Braiek, H., Reid, T. and Khomh, F., 2024, October. In-Simulation Testing of Deep Learning Vision Models in Autonomous Robotic Manipulators. In Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (pp. 2187-2198).*
- *Humeniuk, D., Ben Braiek, H., Khomh, F., Soualy, T. and Reid, T., 2025, December. STRIM: Sim2Real-Aware In-Simulation Testing and Repair of Manipulator Vision Models. In 2026 IEEE Conference on Software Testing, Verification and Validation (ICST) (under revision).*
- *Humeniuk, D., Hamdaqa, M., Ben Braiek, Bennaceur, A. and Khomh, F., 2025, December. Dynasto: Validity-Aware Dynamic–Static Parameter Optimization for Autonomous Driving Testing. In 2026 IEEE Conference on Software Testing, Verification and Validation (ICST) (under revision).*

Workshop publications:

- *Humeniuk, D., Antoniol, G. and Khomh, F., 2022, May. AmbieGen tool at the SBST 2022 Tool Competition. In Proceedings of the 15th Workshop on Search-Based Software Testing (pp. 43-46).*

- *Humeniuk, D., Khomh, F. and Antoniol, G., 2023, May. RIGAA at the SBFT 2023 tool competition-Cyber-physical systems track. In 2023 IEEE/ACM International Workshop on Search-Based and Fuzz Testing (SBFT) (pp. 49-50). IEEE.*

- *Humeniuk, D. and Khomh, F., 2024, April. AmbieGenVAE at the SBFT 2024 Tool Competition-Cyber-Physical Systems Track. In Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing (pp. 45-46).*

- *Humeniuk, D. and Khomh, F., 2024, April. AmbieGen at the SBFT 2024 Tool Competition-CPS-UAV Track. In Proceedings of the 17th ACM/IEEE International Workshop on Search-Based and Fuzz Testing (pp. 69-70).*

APPENDIX B RESULTS OF STATISTICAL SIGNIFICANCE TESTING FOR RQ2 IN RILAST APPROACH EVALUATION

Table B.1 Number of revealed failures and their sparseness in UC1

Number of failures				Sparseness			
A	B	p-value	Effect size	A	B	p-value	Effect size
ran	ga1	0.025	-0.6, L	ran	ga1	0.97	-0.02, N
ran	ga2	0.001	-0.89, L	ran	ga2	0.076	0.48, L
ran	vr_ran	0.005	0.71, L	ran	vr_ran	0.427	-0.22, S
ran	vr_ga1	0.444	-0.21, S	ran	vr_ga1	0.734	0.1, N
ran	vr_ga2	<0.001	-1.0, L	ran	vr_ga2	0.93	0.029, N
ran	vo_ran	<0.001	-1.0, L	ran	vo_ran	0.241	0.32, S
ran	vo_ga1	<0.001	-1.0, L	ran	vo_ga1	0.006	0.74, L
ran	vo_ga2	<0.001	-1.0, L	ran	vo_ga2	0.254	0.286, S
ga1	ga2	0.288	-0.29, S	ga1	ga2	0.031	0.58, L
ga1	vr_ran	0.001	0.87, L	ga1	vr_ran	0.385	-0.24, S
ga1	vr_ga1	0.149	0.39, M	ga1	vr_ga1	0.385	0.24, S
ga1	vr_ga2	0.168	-0.343, M	ga1	vr_ga2	0.539	0.157, S
ga1	vo_ran	0.001	-0.91, L	ga1	vo_ran	0.021	0.62, L
ga1	vo_ga1	0.001	-0.91, L	ga1	vo_ga1	0.003	0.8, L
ga1	vo_ga2	<0.001	-1.0, L	ga1	vo_ga2	0.05	0.486, L
ga2	vr_ran	<0.001	1.0, L	ga2	vr_ran	0.089	-0.46, M
ga2	vr_ga1	0.009	0.7, L	ga2	vr_ga1	0.089	-0.46, M
ga2	vr_ga2	0.953	-0.021, N	ga2	vr_ga2	0.018	-0.586, L
ga2	vo_ran	0.001	-0.91, L	ga2	vo_ran	0.186	-0.36, M
ga2	vo_ga1	0.001	-0.86, L	ga2	vo_ga1	0.97	-0.02, N
ga2	vo_ga2	<0.001	-1.0, L	ga2	vo_ga2	0.135	-0.371, M
vr_ran	vr_ga1	0.002	-0.81, L	vr_ran	vr_ga1	0.273	0.3, S
vr_ran	vr_ga2	<0.001	-1.0, L	vr_ran	vr_ga2	0.364	0.229, S
vr_ran	vo_ran	<0.001	-1.0, L	vr_ran	vo_ran	0.273	0.3, S
vr_ran	vo_ga1	<0.001	-1.0, L	vr_ran	vo_ga1	0.089	0.46, M
vr_ran	vo_ga2	<0.001	-1.0, L	vr_ran	vo_ga2	0.23	0.3, S
vr_ga1	vr_ga2	0.004	-0.714, L	vr_ga1	vr_ga2	0.93	-0.029, N
vr_ga1	vo_ran	<0.001	-1.0, L	vr_ga1	vo_ran	0.064	0.5, L
vr_ga1	vo_ga1	<0.001	-0.96, L	vr_ga1	vo_ga1	0.011	0.68, L
vr_ga1	vo_ga2	<0.001	-1.0, L	vr_ga1	vo_ga2	0.188	0.329, S
vr_ga2	vo_ran	<0.001	-0.893, L	vr_ga2	vo_ran	0.013	0.614, L
vr_ga2	vo_ga1	<0.001	-0.886, L	vr_ga2	vo_ga1	0.001	0.814, L
vr_ga2	vo_ga2	<0.001	-1.0, L	vr_ga2	vo_ga2	0.057	0.429, M
vo_ran	vo_ga1	0.052	-0.52, L	vo_ran	vo_ga1	0.045	0.54, L
vo_ran	vo_ga2	<0.001	-1.0, L	vo_ran	vo_ga2	0.578	-0.143, N
vo_ga1	vo_ga2	0.004	-0.7, L	vo_ga1	vo_ga2	0.015	-0.6, L

Table B.2 Number of revealed failures and their sparseness in UC2

Number of failures				Failure sparseness			
A	B	p-value	Effect size	A	B	p-value	Effect size
ran	ga1	0.343	-0.26, S	ran	ga1	0.345	0.26, S
ran	ga2	0.303	0.257, S	ran	ga2	0.578	0.143, N
ran	vr_ran	0.011	0.65, L	ran	vr_ran	0.448	0.2, S
ran	vr_ga1	0.721	0.113, N	ran	vr_ga1	0.515	-0.2, S
ran	vr_ga2	<0.001	-0.98, L	ran	vr_ga2	1	0.0, N
ran	vo_ran	<0.001	-1.0, L	ran	vo_ran	0.212	-0.34, M
ran	vo_ga1	<0.001	-1.0, L	ran	vo_ga1	0.275	-0.291, S
ran	vo_ga2	<0.001	-1.0, L	ran	vo_ga2	0.473	0.2, S
ga1	ga2	0.048	0.436, M	ga1	ga2	0.93	-0.029, N
ga1	vr_ran	0.004	0.725, L	ga1	vr_ran	0.767	-0.083, N
ga1	vr_ga1	0.476	0.212, S	ga1	vr_ga1	0.173	-0.4, M
ga1	vr_ga2	0.001	-0.88, L	ga1	vr_ga2	0.521	-0.18, S
ga1	vo_ran	<0.001	-1.0, L	ga1	vo_ran	0.054	-0.52, L
ga1	vo_ga1	<0.001	-1.0, L	ga1	vo_ga1	0.084	-0.455, M
ga1	vo_ga2	<0.001	-1.0, L	ga1	vo_ga2	0.623	-0.14, N
ga2	vr_ran	0.044	0.47, M	ga2	vr_ran	0.939	-0.024, N
ga2	vr_ga1	0.656	-0.125, N	ga2	vr_ga1	0.33	-0.268, S
ga2	vr_ga2	<0.001	-0.993, L	ga2	vr_ga2	0.838	-0.057, N
ga2	vo_ran	<0.001	-1.0, L	ga2	vo_ran	0.084	-0.429, M
ga2	vo_ga1	<0.001	-1.0, L	ga2	vo_ga1	0.198	-0.312, S
ga2	vo_ga2	<0.001	-1.0, L	ga2	vo_ga2	0.93	-0.029, N
vr_ran	vr_ga1	0.03	-0.594, L	vr_ran	vr_ga1	0.157	-0.396, M
vr_ran	vr_ga2	<0.001	-1.0, L	vr_ran	vr_ga2	0.767	-0.083, N
vr_ran	vo_ran	<0.001	-1.0, L	vr_ran	vo_ran	0.07	-0.467, M
vr_ran	vo_ga1	<0.001	-1.0, L	vr_ran	vo_ga1	0.117	-0.394, M
vr_ran	vo_ga2	<0.001	-1.0, L	vr_ran	vo_ga2	0.974	-0.017, N
vr_ga1	vr_ga2	0.001	-0.95, L	vr_ga1	vr_ga2	0.36	0.275, S
vr_ga1	vo_ran	<0.001	-1.0, L	vr_ga1	vo_ran	0.573	-0.175, S
vr_ga1	vo_ga1	<0.001	-1.0, L	vr_ga1	vo_ga1	0.6	-0.159, S
vr_ga1	vo_ga2	<0.001	-1.0, L	vr_ga1	vo_ga2	0.173	0.4, M
vr_ga2	vo_ran	<0.001	-1.0, L	vr_ga2	vo_ran	0.14	-0.4, M
vr_ga2	vo_ga1	<0.001	-1.0, L	vr_ga2	vo_ga1	0.275	-0.291, S
vr_ga2	vo_ga2	<0.001	-1.0, L	vr_ga2	vo_ga2	0.734	0.1, N
vo_ran	vo_ga1	0.377	-0.236, S	vo_ran	vo_ga1	0.751	0.091, N
vo_ran	vo_ga2	0.005	-0.76, L	vo_ran	vo_ga2	0.021	0.62, L
vo_ga1	vo_ga2	0.034	-0.555, L	vo_ga1	vo_ga2	0.073	0.473, M